1993

An object-oriented logistics over the shore simulation: an aid in throughput estimation

Noel, Jack S. II

Monterey, California. Naval Postgraduate School

NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

AN OBJECT-ORIENTED LOGISTICS OVER THE SHORE SIMULATION: AN AID IN THROUGHPUT ESTIMATION

by

Jack S. Noel II

September 1993

Thesis Advisor: William G. Kemple

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704-0188

1.AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE 1993 September	3.REPORT TYPE AND DATES COVERED Master's Thesis

4.TITLE AND SUBTITLE AN OBJECT-ORIENTED LOGISTICS OVER THE SHORE SIMULATION: AN AID IN THROUGHPUT ESTIMATION

5.FUNDING NUMBERS

6.AUTHOR(S) Jack S. Noel II

7.PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Naval Postgraduate School
Monterey CA 93943-5000

8.PERFORMING ORGANIZATION REPORT NUMBER

9.SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10.SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release, distribution is unlimited.

12b. DISTRIBUTION CODE

13.ABSTRACT *(maximum 200 words)*

This thesis documents the design, validation, and demonstration of a simulation model for the instream offload of vehicles from a Roll On/Roll Off ship. The model is an object-oriented, discrete event simulation written in MODSIM II. The objective is to design and demonstrate a model that can accurately estimate throughput times for the total offload of a vessel instream using various mixes of lighterage. With this tool, Logistics Over The Shore (LOTS) planners will be better able to estimate throughput and possibly tailor their mix of lighterage to a given set of fixed parameters.

14. SUBJECT TERMS Object-Oriented, LOTS, JLOTS, Logistics, Simulation, MODSIM	15.NUMBER OF PAGES 252
	16 PRICE CODE

17.SECURITY CLASSIFICATION OF REPORT	18.SECURITY CLASSIFICATION OF THIS PAGE	19.SECURITY CLASSIFICATION OF ABSTRACT	20.LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

Approved for public release; distribution is unlimited.

An Object-Oriented Logistics Over The Shore Simulation: An Aid in Throughput Estimation

by

Jack S. Noel II
Lieutenant, United States Navy
B.S., United States Naval Academy, 1985

Submitted in partial fulfillment of the
Requirements for the degree of

MASTER OF SCIENCE IN OPERATIONS RESEARCH

from the

NAVAL POSTGRADUATE SCHOOL
September 1993

Author: [redacted]
Jack S. Noel II

Approved By: [redacted]
William G. Kemple, Thesis Advisor

[redacted]
Keebom Kang, Second Reader

[redacted]
Peter Purdue, Chairman
Department of Operations Analysis

ABSTRACT

This thesis documents the design, validation, and demonstration of a simulation model for the instream offload of vehicles from a Roll On/Roll Off ship. The model is an object-oriented, discrete event simulation written in MODSIM II. The objective is to design and demonstrate a model that can accurately estimate throughput times for the total offload of a vessel instream using various mixes of lighterage. With this tool, Logistics Over The Shore (LOTS) planners will be better able to estimate throughput and possibly tailor their mix of lighterage to a given set of fixed parameters.

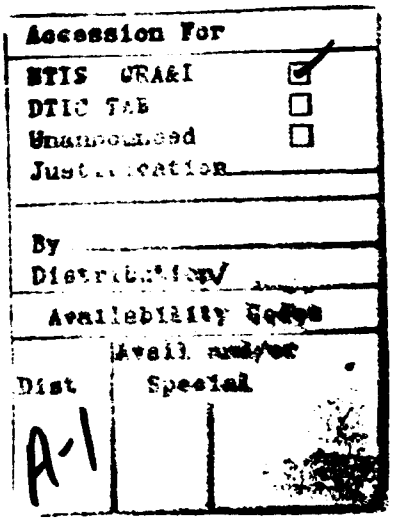

TABLE OF CONTENTS

EXECUTIVE SUMMARY

Responding to the collapse of the Soviet Union, the U.S. military is shifting its focus from global war and land combat in Europe to regional contingencies in the third world. This new role of the U.S. military will require more flexibility and speed than has ever been required previously.

Responding to the potential requirements to conduct contingency operations in highly varied geophysical and meteorological conditions, the DoD has developed the Logistics Over The Shore (LOTS) system as an alternative to the modern port.

LOTS is an integrated system of equipment, personnel, and procedures used to load and unload ships without the benefit of fixed port facilities, in either friendly or undefended territory. It is designed to provide the flexibility U.S. forces will need in the austere environments that they are likely to encounter. LOTS operations may be conducted over unimproved shorelines, through fixed

ports that are not accessible to deep draft shipping, and through fixed ports that simply lack the facilities for efficient offload without LOTS capabilities. The most involved form of LOTS, and the focus for this thesis, is the offload of equipment and cargo over an unimproved shoreline, or instream offload.

A basic problem facing military planners is that they currently have no comprehensive means of estimating the throughput capability of an operation for various mixes of oceangoing ships, lighters, and material handling equipment.

The objective of this thesis was to build an Object-Oriented computer simulation model in MODSIM II that estimates the throughput capability of an instream vehicle discharge operation from a RO/RO type vessel. The model used to generate these planning factors serves as a computer-based decision aid wherein the input can be modified to allow a planner to experiment with various combinations of equipment and shipping configurations.

To ensure that the simulation model performed as required, a two phase process was used to first validate, and

then demonstrate the model. The validation phase consisted of running the model using a real world scenario for which the empirical offload time was known, and then comparing the two. After 300 replications, the mean offload time from the model compares well with the empirical offload time for the same scenario.

The second phase was to demonstrate the model. A four point design space was devised where the variables were a high and low mix of lighters at short and long distances from the beach. As would expected be when comparing the results of these four scenarios, the high mix of lighters had a significantly shorter offload time than the low mix, and the short distance scenarios had shorter offload times than the long distance scenarios.

In summary, the model is valid for estimating RO/RO in-stream offload times within the confines of the assumptions made in the modeling process.

I. INTRODUCTION

A. THE IMPORTANCE OF LOTS

Responding to the collapse of the Soviet Union, the U.S. military is shifting its focus from global war and land combat in Europe to regional contingencies in the third world. Force sizes are being reduced and fewer units will be forward deployed. The U.S. will depend heavily on airlift and sealift to achieve rapid response. This new role of the U.S. military will require more flexibility and speed than has ever been required previously. For its part, the U.S. Navy has shifted focus towards littoral warfare, and the ability to project military power in the worlds coastal regions is gaining in importance.

Military and commercial airlift have been relied upon heavily in the past to achieve a rapid build up of forces and will remain a key asset. Airlift alone has never been able to transport more than a small fraction of the required assets, however, and with fewer land-based, forward deployed forces, the demand for airlift will be far too large for a

reasonable number of aircraft to accommodate. Thus, sea lift will play an ever increasing part.

The use of strategic sealift in the rapid deployment, sustainment and re-deployment of forces overseas is essential in the execution of any US Department of Defense (DoD) contingency operation. Due to the expense of acquiring and maintaining large fleets of lift assets, DoD has turned increasingly to the commercial sector to provide lift. The positive aspect of this is increased cost savings for all concerned, provided there is no actual contingency. One negative aspect is that commercial shipping has grown dependent on fully cellularized containerships. The large modern port facilities normally required to offload commercial vessels limits the flexibility of our military forces. In the regional contingencies that are likely to challenge U.S. forces in the future, flexibility is essential.

Responding to the potential requirements to conduct contingency operations in highly varied geophysical and meteorological conditions, the DoD has developed the

Logistics Over The Shore (LOTS) system as an alternative to the modern port.

B. GENERAL DESCRIPTION OF A LOTS OPERATION

LOTS is an integrated system of equipment, personnel, and procedures used to load and unload ships without the benefit of fixed port facilities, in either friendly or undefended territory. It is designed to provide the flexibility U.S. forces will need in the austere environments that they are likely to encounter. LOTS operations may be conducted over unimproved shorelines, through fixed ports that are not accessible to deep draft shipping, and through fixed ports that simply lack the facilities for efficient offload without LOTS capabilities. The most involved form of LOTS, and the focus for this thesis, is the offload of equipment and cargo over an unimproved shoreline, or instream offload.

The LOTS transfer operation can be broken down into three areas; the offload of roll-on/roll-off (RO/RO) ships through a RO/RO discharge facility (RRDF), the offload and transfer of containers from either a self-sustaining or a

non-self-sustaining containership, and the transfer of fuel from an Offshore Petroleum Discharge System (OPDS). The first two operations are the major concern regarding throughput. For the third operation, once the OPDS is in place, the throughput is a known quantity and is easily controlled.

C. PROBLEM

A basic problem facing military planners and LOTS operational commanders is that they currently have no comprehensive means of estimating the throughput capability of an operation for various mixes of oceangoing ships, lighters, and material handling equipment. The joint tactical publication regarding LOTS [Ref. 1] provides a limited set of planning factors that would be useful to planners provided that their mix of equipment falls within the limited scope of these factors.

Conversations with personnel at the JLOTS Test Directorate [Ref. 2] and personnel in the Logistics Directorate of the Joint Staff [Ref. 3] revealed that there is a pressing need for a comprehensive set of planning factors as

well as some sort of tool (e.g., a computer simulation model) that can be used by planners in the future to update these planning factors as required.

Several computer based simulation models of LOTS operations have been developed, and the JLOTS Test Directorate has some of these models in hand, but each of these simulations is written in a different language, with its own software and hardware requirements. The JLOTS Test Directorate has found them difficult to comprehend and impossible to implement as an actual functioning tool.

D. PROPOSAL

The limited set of planning factors currently available in *Joint Pub 4-01.6* [Ref. 1] is inadequate for the needs of JLOTS planners. The objective of this thesis is to build a computer simulation model that estimates the throughput capability of an instream vehicle discharge operation from a RO/RO type vessel through an RRDF under various scenarios. The model used to generate these planning factors will serve as a computer-based decision aid wherein the input can be modified to allow a planner to experiment with various

combinations of equipment and shipping configurations. As equipment configurations change, and the availability of lighterages varies, the model can easily be modified to reflect these updates.

Validation of the model will be accomplished by compairing model results with real world figures for a known scenario. Further, we will use four scenarios to illustrate the capability of the model. These scenarios reflect a range of situations that a LOTS planner may face and will serve as instruction as to how the model may be employed.

This thesis is organized as follows:

Chapter	Title/Description
I.	INTRODUCTION.
II.	THE LOTS OPERATION. This chapter provides a brief description of a LOTS operation instream offload, a description of the typical environment, and a description of the equipment used in the operation.
III.	THE MODEL. This chapter provides a complete description of the simulation model including assumptions and input and output data.
IV.	ANALYTICAL PROCEDURES. This chapter provides the origin for model event times and a discussion of the results.
V.	CONCLUSIONS. This chapter describes conclusions and recommendations.

II. THE LOGISTICS OVER THE SHORE OPERATION

A. OVERVIEW OF THE LOTS OPERATION

Logistics over the shore involves loading and unloading military or commercial ships in what would be considered less than ideal circumstances. The operation could take place instream or in port facilities that are either damaged or in some other way lack the facilities typically required to handle a modern ship. In the instream offload, the ships are located anywhere from one to several miles offshore. The cargo and vehicles are then loaded onto various types of lighterage for further transfer over the beach to marshaling areas. The vehicles and cargo then continue their journey on land via the standard means. A LOTS system is composed of the following basic components: [Ref. 1]

1. Seagoing cargo vessel
2. Off shore cargo discharge facility
3. Shallow draft lighterage
4. Shoreside discharge facility

The study of a LOTS operation can be subdivided into four major areas; vehicle offload from a RO/RO type vessel, container offload from either a self-sustaining or non-self-sustaining container ship, break bulk offload, and the bulk offload of liquids such as fuel and water. This thesis is focused on the offload of vehicles from a RO/RO type vessel, thus, the remainder of this chapter will be limited to descriptions of this operation and the specific equipment involved.

A typical RO/RO LOTS operation can best be illustrated by describing what happens to the cargo as it moves from the bowels of the seagoing vessel to the shoreside discharge site. Figure 1 depicts the setup of some of the typical LOTS components [Ref. 1]. On the beach side of the operation, discharge of cargo can be accomplished on a bare beach, on an Elevated Causeway (ELCAS) or pier discharge which could be either a floating causeway pier or the pier in an unimproved port. Some of the different types of vessels that can be offloaded using the LOTS system are also depicted in Figure 1. The Non-self-sustaining container (NSSC) ship would be offloaded using a crane ship (T-ACS),

the Roll On/Roll Off ship has a RO/RO Discharge Facility lashed alongside, and the Maritime Prepositioned Ship (MPS) is being offloaded using Lift On/Lift Off or LO/LO techniques as well as providing fuel and water to the beach.

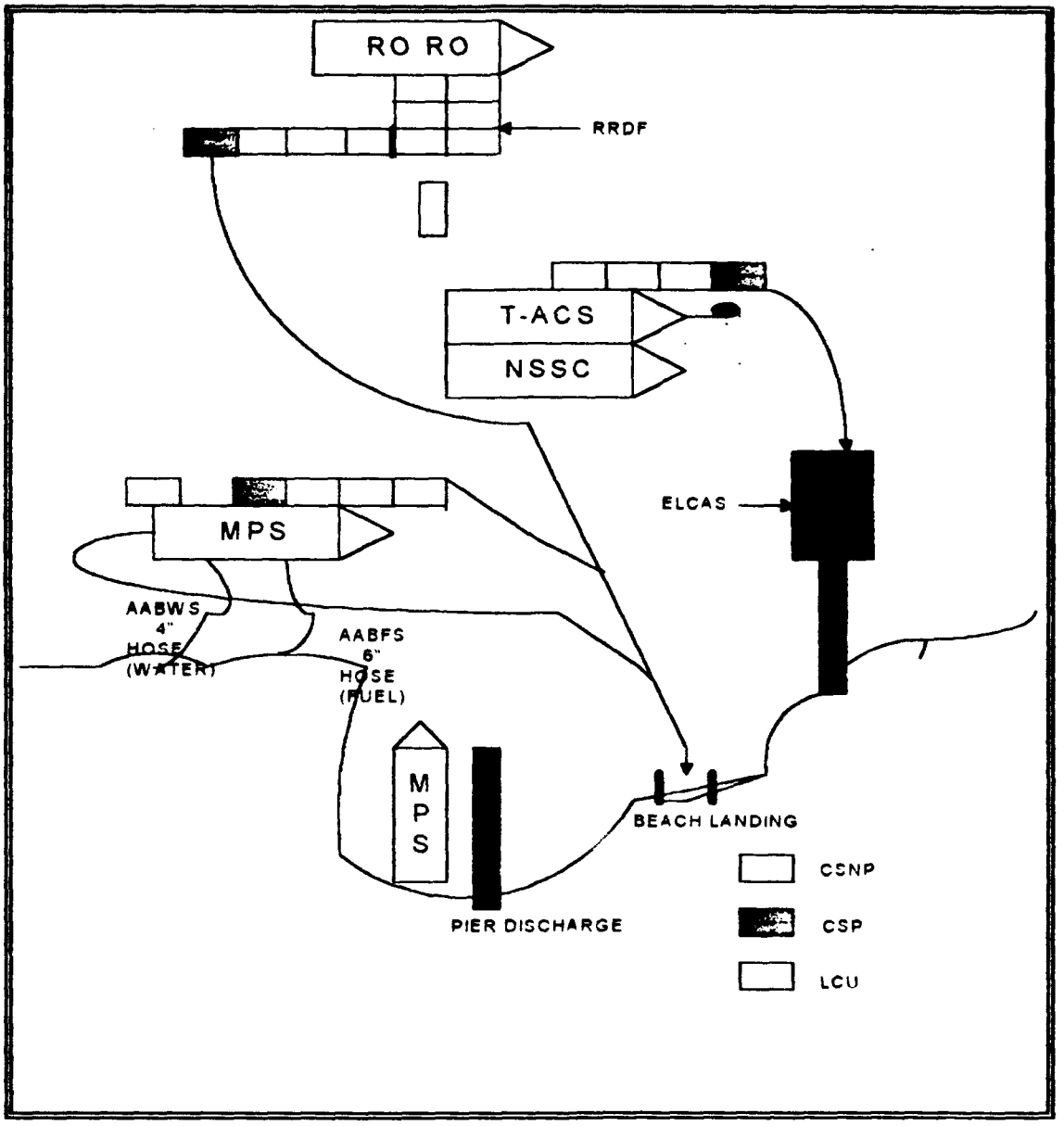

Figure 1: The LOTS Operation

The offload of a RO/RO can be accomplished in two ways. The first method is LO/LO operations in which vehicles and cargo are craned off, either by the ships organic crane or by a T-ACS crane ship. The second method is to use a RO/RO Discharge Facility (RRDF) and simply drive the vehicles onto the lighters. The lighters wait in queues near the ship for an open discharge station. When a discharge station becomes available the Ship Lighterage Control Point (SLCP) directs a lighter alongside. The cargo or rolling stock is then loaded aboard and the lighter casts off from the discharge point to transit to the shoreside queue. At the shoreside queue the lighter is directed by the Beach Lighterage Control Point (BLCP) to the first available discharge point. The cargo is then discharged and proceeds to the marshaling area. The lighter then departs the discharge point and control is transferred from the BLCP to the appropriate SLCP which will tell the lighter where to pick up its next load. Several components exist to perform each of the tasks listed above. They can operate in many combinations to perform the basic LOTS functions. A description of the components

available and how they integrate into the overall system follows.

B. ROLL ON/ROLL OFF SHIP

There are many classes of Roll On/Roll Off ships, but they can be divided into two basic types for this analysis: self-sustaining and non-self-sustaining. The major difference is that a self-sustaining ship has its own ramp for vehicle loading and discharge. In typical commercial operations a self-sustaining RO/RO would moor and lower its vehicle ramp directly onto the pier to commence the offload of vehicles. A ramp provided from port facilities would be married up to a non-self-sustaining ship. In the instream offload, an RRDF is assembled and moored alongside the ship, and in the case of the self-sustaining vessel, the ships ramp is then lowered onto the RRDF platform. In the case of a non-self-sustaining ship, a 120 foot offloading ramp is added to the RRDF. [Ref.1]

The introduction of sea state and current can significantly affect the throughput capability of an offload. For an offload using an RRDF, operation is limited to sea state

two and no more than four knots of current. The discharge of vehicles can continue using LO/LO operations if the sea state exceeds the RRDF's parameters somewhat. But LO/LO operations are also limited in more challenging sea states depending on the equipment and the ability of the crew. In any case LO/LO operations are limited to sea states no higher than three. [Ref. 4]

C. LIGHTERAGE

1. LCU 1466, 1610, and 2000 Class

The LCU is a conventional displacement vessel capable of transporting containers, breakbulk cargo, outsized cargo, vehicles and personnel from the ship to the shoreside discharge point. The 1466 and 1610 class are self-sustaining once deployed to the theater in the sense that they are fully equipped to support their crew once they are delivered to the area of operations. The LCU 2000 class is both self-deployable and self-sustaining. All three classes are therefore capable of extended missions with endurance based upon provisions and fuel capacity. All three LCU classes are equipped with twin screws and a stern anchor, so they can beach and retract under their own power. Cargo may also be discharged from LCU's at a floating causeway pier or

an elevated causeway pier (ELCAS). For discharge at the beach or to a floating causeway pier, rolling stock is driven or towed off over the bow ramp of the LCU. If discharge is accomplished at an ELCAS the cargo is craned off. The characteristics of the three classes of LCU are listed in Table 1. [Ref. 1]

TABLE 1. LCU CHARACTERISTICS

	1466 CLASS	1610 CLASS	2000 CLASS
CARGO CAPACITY PAYLOAD	187 ST	187 ST	188 ST
SPEED MAX	8.0 kts	12 kts	12 kts
FULL LOAD	6.5 kts	11 kts	10 kts
RANGE	1200 nm @ 6 kts	1200 nm @ 6 kts	4500 nm @11.5 kts
FUEL CAPACITY	3542 GALS	3290 GALS	92000 GALS
BURN RATE	34 GPH	36 GPH	41.6 GPH
DRAFT (LOADED) FWD	3'	3' 2"	4'
AFT	4'	6' 5"	9'

2. Logistics Support Vessel (LSV)

The Logistics Support Vessel is a large conventional displacement watercraft capable of transporting large amounts of cargo to almost any port in the world. Much like

a large LCU, the LSV can carry loads such as 11 M1 tanks, 21 M2 Infantry Fighting Vehicles, or 48 20-foot containers. The LSV's are both self-sustaining and self-deployable. LSV's are capable of beaching and retracting under their own power, thus, providing the most basic means of discharge. Cargo can also be discharged using a floating causeway pier or an ELCAS. As with an LCU, the rolling stock is driven or towed off in the first two cases, and craned off in the third. The characteristics of the LSV are listed in Table 2. [Ref. 5]

TABLE 2. LSV CHARACTERISTICS

CARGO CAPACITY	
PAYLOAD	2000 ST
SPEED MAX	11.6 kts
FULL LOAD	10.0 kts
RANGE	6500 nm @ 11 kts
FUEL	
CAPACITY	165,000 GALS
BURN RATE	146 GPH
DRAFT (LOADED)	
FWD	6'
AFT	6'

3. Causeway Ferry (CWF)

A causeway ferry (CWF) is assembled from Navy standard 90 x 21 foot causeway sections. From one to three *causeway section's non-powered* (CSNP) can be coupled with a *causeway section powered* (CSP) to form a ferry. The CSP contains two waterjet propulsion assemblies that are capable of propelling the loaded ferry from the ship to the shoreside discharge point. The CWF is capable of beaching and retracting under its own power as well as discharging at an ELCAS using a crane. The causeway ferry is capable of operating a full 10 hour shift without refueling. The causeway sections are easily loaded aboard several types of ships, and they are the basis for several major LOTS components such as the Floating Causeway Pier and the Side Loadable Warping Tug in addition to the CWF. Figure 2 shows the various configurations for a CWF. [Ref. 4]

D. RO/RO DISCHARGE FACILITY (RRDF)

The RRDF provides an interface between the sea going vessels and the lighters for the offload of vehicles. The RRDF itself is configured from six CSNP's in a three-wide by

two-long configuration to provide a platform for the offload ramp, which may be either the ship's ramp for a self-sustaining RO/RO, or part of the RRDF for a non-self-sustaining RO/RO. Vehicles can be driven directly off of the ships, onto the platform and onto either a causeway ferry, an LCU, or an LSV. The RRDF is moored directly to the ship and is helped maintaining its position by a Side Loadable Warping Tug. Assembly and mooring of the RRDF is limited to sea state 0-1 for a non-self-sustaining vessel and from 0-2 for a self-sustaining vessel. The RRDF can, however, be operated safely through sea state 2 in either case. [Ref. 1]

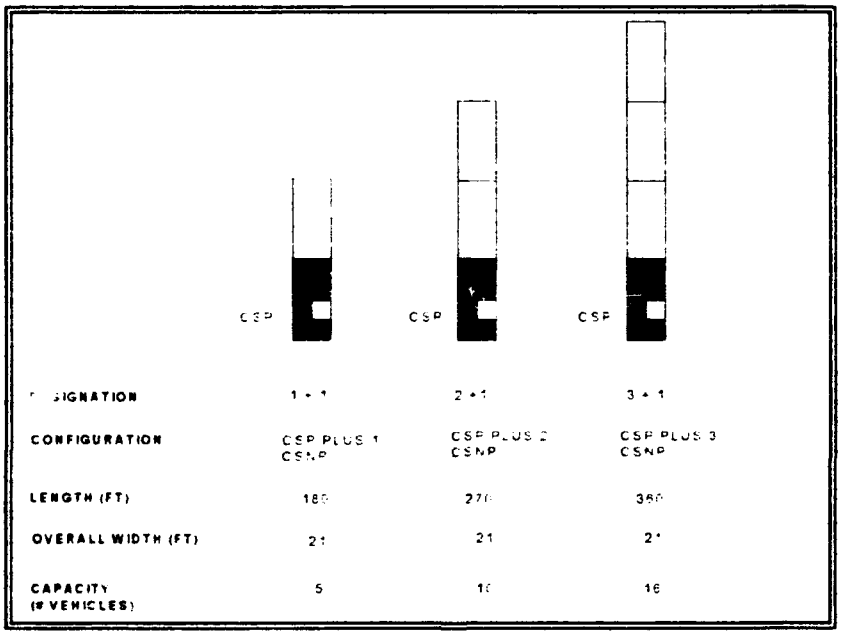

Figure 2: Causeway Ferry Configurations

E. BEACH DISCHARGE FACILITIES

Several types of beach discharge methods may be employed in LOTS operations. The use of one system over another depends on the scenario at hand, the type of ship being offloaded, the lighterage being employed, and most importantly, weather and surf conditions in the discharge area. For the offload of a RO/RO type vessel, using causeway ferries, LCU's, or LSV's, there are three basic beach discharge methods: bare beach operations, causeway pier operations, and elevated causeway pier operations. The factors that influence the choice of each are listed below.

1. Bare Beach Operations

Throughput during bare beach operations are primarily dependent on beach gradient and characteristics, weather, wave height, and the beach consistency. The type of cargo to be offloaded is also of concern since offloading is done in the surf zone. The possibility of vehicles stalling or being unable to gain traction is viable and must be considered. In the typical offload of RO/RO type cargo the vehicles would simply be driven or towed off of the

lighter onto the beach where they would then be directed to a staging area. [Ref. 1]

2. Floating Causeway Pier Operations

A Floating Causeway Pier would normally be assembled in the amphibious assault phase of an operation and remain behind for the subsequent offload of the assault follow-on echelon until the more permanent ELCAS could be installed. Although RO/RO cargo is typically discharged directly to the beach, the floating causeway pier provides a means for offloading a safe distance from the surf zone if it is required. A floating causeway pier is composed of 1 CSNP (beach end configured), and 1 CSNP (sea end configured) with the required number of CSNP's in between to meet the desired depth at the sea end. Floating causeway piers are capable of operating in wave height of 4 feet and a lateral wind force of up to 40 knots with a 3 knot current.

3. Elevated Causeway Operations

The ELCAS allows containers, break bulk cargo and vehicles to be discharged without contending with the surf zone. ELCAS is a rapidly installable pier facility that can be extended up to 3000 feet beyond the surf zone to provide

mooring for any type of military lighter or commercial barge. The amount of roadway actually installed depends on what is required to meet the 12 foot depth requirement at the pierhead. The pierhead of the ELCAS is double width (72 feet) and equipped with two air-bearing turn-tables and two 180-ton cranes. The ELCAS would be installed with the arrival of the first container ship. RO/RO cargo is normally discharged on a bare beach, but can be discharged at the ELCAS if the vehicles are within the weight limits of the crane used on the ELCAS. Weight limits preclude the offload of such vehicles as tanks and large, heavily loaded trucks.

This equipment and its characteristics discussed in the preceding sections must be faithfully modeled in the simulation to ensure that the output is reasonable and reliable. The following chapter is a thorough description of the RO/RO Offload model and how this was accomplished.

III. THE MODEL

A. BACKGROUND

We develop the RO/RO Offload simulation model to analyze throughput for the RO/RO portion of an instream LOTS operation. The model was written in MODSIM II, an Object-Oriented simulation language. MODSIM II "is a general-purpose, modular, block-structured high-level programming language which provides direct support for object-oriented programming and discrete-event simulation." [Ref. 6:p. 1] The prime elements of the RO/RO Offload model are created as objects.

> Objects in MODSIM are dynamically allocated data structures coupled with routines, called methods. The fields in the objects data structure define its state at any instant in time while its methods describe the actions which the object can perform. [Ref. 6:p. 103]

As an example, an LCU 2000 class lighter is modeled as a lighter object, possessing the attributes that are unique to that class such as cargo capacity, speed, fuel capacity, and fuel burn rate. The fuel on board the individual lighter decrements as time passes by a method within the lighter object called ***BurnFuel***. All of the objects within the

simulation interact to pass time in a realistic fashion so that statistical data may be gathered for later analysis.

We model all of the components necessary to perform the RO/RO portion of an instream LOTS operation. The objects are generic enough so that any changes in the number and type of ships, beaches, or lighterage, can be modified in the model by simply changing the input files. The key objects in the RO/RO Offload model are described in the sections that follow. Figure 3 is a pictorial representation of the objects in the model which correspond to the following descriptions.

B. ASSUMPTIONS

Several assumptions were required in order to define and narrow the scope of the simulation model. The assumptions are:

1. Weather is not a factor in the simulated offloads. In real world operations LOTS is limited in practice to sea states of two or below.
2. One ship is offloaded at a time.
3. All lighters are dedicated to the offload of one ship.
4. There are no breakdowns in equipment.

5. Serials are prepared aboard the ship prior to the arrival of the lighter. In other words, the vehicles are standing by for onload when a lighter arrives.

6. Lighters can perform operaterations simultaneously. More than one lighter can onload at the same time, several lighters may transit simultaneously, and so on.

C. SHIPS

As discussed in Chapter II of this thesis, the oceangoing vessels that transport vehicles to the offload area in a LOTS operation are RO/RO's. The object within the RO/RO Offload model that represents these vessels is the ***RoRoObj***. The actual movement of the RO/RO to the Area of Operations (AOR) is not modeled in the simulation because the question to be answered is throughput in the offload phase, therefore, simulation starts with the ships on station. The dominant effect the RO/RO has in this phase is the number of spots the vessel has available for onload to a particular type of lighter, and whether or not the RO/RO is self-sustaining. Additionally, some classes of ships are partitioned internally so that portions of the cargo can only be handled by a specific spot, which limits the number and type of lighters available to offload that cargo. These aspects

of a particular class of RO/RO can be modeled simply by changing the input files for each vessel desired in a particular scenario.

The ***RoRoObj*** possesses a method called ***MakeLoad*** which actually constructs the serials based on the maximum capacity of a given lighter. The number and type of spots for a ship are input variables. For example, a RO/RO configured with an RRDF and one organic crane has one LO/LO spot, one CWF spot on the RRDF and one LCU spot on the RRDF. The LO/LO spot is usually on the opposite side of the vessel from the RRDF, and can accept both CWF's and LCU's. As an example of a partitioned load, the aft portion of the *Algol (SL-7)* class Vehicle Cargo Ships can only be offloaded by LO/LO operations and can contain up to 60 vehicles. The ***MakeLoad*** method keeps track of these 60 vehicles and ensures that they decrement appropriately when onload of a lighter is conducted at the LO/LO spot.

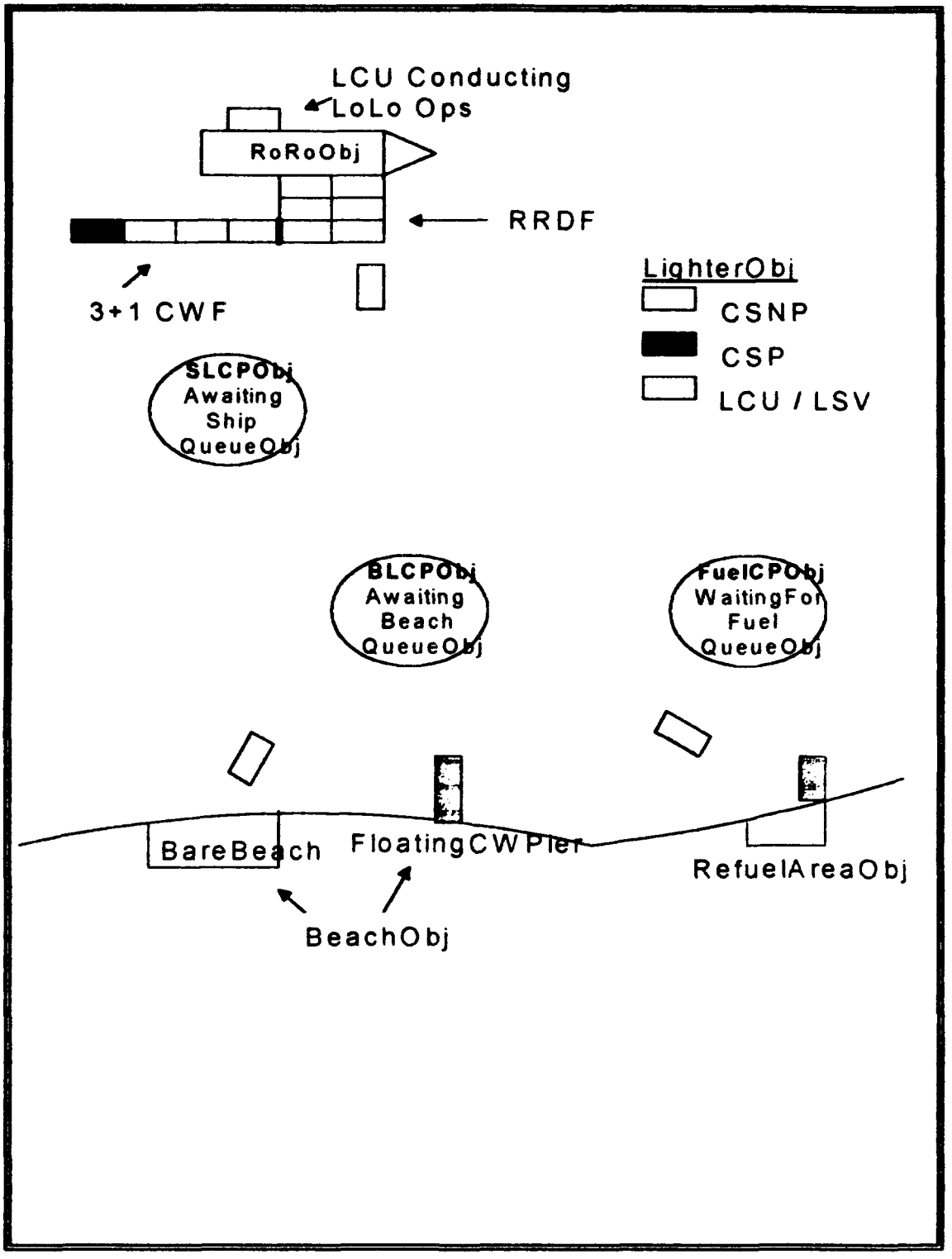

Figure 3: The Object-Oriented Approach to LOTS

D. LIGHTERAGE CONTROL POINTS

The Lighterage Control Points serve as the interface between the queue in which the lighters wait for an available spot at a particular location, and that location. For example, the Ship Lighterage Control Point (SLCP) interfaces between the lighters waiting for a spot at the ship, and the ship itself. There are three Lighterage Control Points modeled in the RO/RO Offload model. In addition to the ***SLCPObj***, the ***BLCPObj*** interfaces with the Beach objects, the ***FuelCPObj*** interfaces with the refueling area objects, and, each interfaces with it's respective queue. Each of the modules in which the Lighterage Control Point objects are defined also has the definition for the associated queue object, such as the ***AwaitingShipQueueObj***. The queue objects actually hold the lighter objects until a spot opens up at the associated location.

E. BEACH AREAS

Typically the offload of rolling stock in the beach area consists of starting the vehicles up and driving them away. The one exception to this rule is when an ELCAS is in use

and vehicles are lifted from the lighters. The RO/RO Offload model provides for three beach types: bare beach, floating causeway pier, and ELCAS. Each instance of a ***BeachObj*** in the model, however, only varies in the number of spots available for lighter offload. The spots are not unique to a particular type of lighter, but if this attribute were desired it would entail only a minor modification to the source code. The ***BeachObj*** has within it a method which keeps track of the time that the spots lay idle for later analysis.

F. REFUEL AREA

The refueling area in a LOTS operation is typically a shore site equipped to refuel the lighters. Each of the lighters is normally capable of operating for a minimum of a ten hour shift without refueling, but the time to transit to the refueling area and to refuel can add a significant amount of variance when looking at throughput.

The refueling area, modeled as a ***RefuelAreaObj***, has two attributes which affect simulation time; the number of spots available for refueling and the pump rate. Both of these

attributes can be modified in the input files. One other factor affecting the refueling operation is the distance of the refueling area with respect to both the ship and the beach. These are also input parameters.

G. LIGHTERS

The ***LighterObj*** is the key object in the RO/RO Offload model in that it alone contains the methods which control the passage of simulation time for each of the events modeled. Each instance of a lighter object has its unique attributes such as speed and cargo capacity that will come into play when the time required for a given event to pass is calculated.

The basic lighter cycle consists of eight events, each of which is associated with a method in the LighterObj to control the passage of simulation time. During each of these events, as well as the time spent in the queues, the lighter burns fuel. The eight events in the basic lighter cycle are described below. A pictorial description using object-oriented simulation pictures (OOS-Pics) [Ref. 7] can be found in Appendix A.

1. Approach and Moor to Ship

At the beginning of simulation all lighters are in standby in the ***AwitingShipQueue***. When a ship has a free spot, it asks the ***SLCPObj*** to remove a lighter requiring the appropriate spot type from the queue. The ***SLCPObj*** then tells the lighter to Approach And Moor, thus initializing the lighter cycle. The approach and moor time is calculated, the simulation time is allowed to pass, and fuel is burned for that period of time.

2. Onload

After the lighter completes the approach and moor event, it asks the ship to ***MakeLoad***, which causes the ***RoRoObj*** to create a load and decrement the total onboard the ship appropriately. The ship next tells the lighter to onload. The ***LighterObj*** calculates the onload time, passes the correct amount of simulation time, burns fuel, and tells itself, the lighter, to cast off and clear the ship.

3. Cast and Clear the Ship

The cast and clear event is similar to the other two described above in that the cast and clear time is calculated and allowed to pass, and fuel is burned. After the

lighter completes the cast and clear event it transits to the beach area queue. The ship is next asked to clear the spot just used. This frees it for use by another lighter and starts that lighter's cycle.

4. Transit to the Beach Area Queue

As a lighter transits to the beach area it asks the ***BLCPObj*** to get a beach spot for offloading. The ***BLCPObj*** in turn asks each of the beach objects in the scenario if there are any spots free. If there is a free spot, the lighter is told to approach and moor to the appropriate ***BeachObj***. If there are no spots available the lighter is added to the ***AwaitingBeachQueue***.

The ***TransitTo*** method in the model calculates the transit time based only on speed and distance. This is different from the other events in the cycle where the event times are drawn from random number streams.

5. Approach and Moor to the Beach

Approach and moor at the beach is identical to that on the ship with the exception that the lighters spot requirement is not checked, any lighter type can moor to any beach type. If a spot becomes free at the beach and a

lighter does not happen to be in transit and requesting a spot at that exact moment, the beach asks the ***BLCPObj*** to remove a lighter from the queue. The ***BLCPObj*** then tells the newly removed lighter to approach and moor.

6. Offload

Upon completion of the approach and moor event the lighter is told to offload. Offload time is calculated, simulation time is allowed to pass, and fuel is burned. After completion of the offload, the fuel status of the lighter is checked. If the lighter is at or below the specified minimum fuel percentage it is told to cast and clear the beach and transit to the refueling area. If the fuel status is above the minimum the lighter is told to cast and clear the beach and transit to the ship area. These two events are the last two of the eight in the basic lighter cycle. They are essentially the same as the like named events above, so a detailed description is not required.

7. Refueling

The basic sequence of events is modified slightly if a lighter requires fuel. As mentioned above, after a lighter completes its offload, a check is conducted to

determine if the lighter is at or below its minimum fuel percentage. If the lighter is low on fuel it casts and clears the beach and transits to the refueling area. On the way to the refueling area the lighter checks in with the ***FuelCPObj***, which asks the ***RefuelAreaObj*** to check for empty spots. If there is a spot available the lighter is told to approach and moor. If not, the lighter is added to the ***FuelAreaQueue*** and waits for a spot to open up. After a lighter completes the approach and moor event it is told to refuel. Refuel time is calculated based on the fuel required to fill the lighter to capacity and the pump rate of the ***RefuelAreaObj***. When refueling is complete, the lighter is told to cast and clear and transit to the ship area. The spot in the ***RefuelAreaObj*** is made free and the ***FuelCPObj*** is asked to remove the next lighter from the queue.

After the lighter arrives back in the ship area queue, the cycle starts anew with the lighter waiting for a spot to become free on the ship. Each instance of a LighterObj conducts these events until the offload of every RO/RO in the simulation is complete.

H. MODEL EXECUTION

There are four phases in the execution of the RO/RO Offload simulation. A brief description of these phases and the objects involved in their execution follows.

1. Input

The information required to create a desired scenario is stored in eight ASCII files. These files are easy to create and edit. The ***ListMasterObj*** has the task of creating the necessary objects to read these files and to call the methods that actually read the data. The information for each object to be built is stored in a record, and then added to a ***QueueListObj***. Examples of program input files can be found in Appendix D.

2. Object Building

The ***ObjectBuilderObj*** creates each instance of the objects required for the scenario. It removes the records from the QueueList objects mentioned above, and fires a method in each object that fills that objects fields with the information from the record. The end result of this phase is that all of the objects required to run the desired scenario are built and standing by for replication. All of

the lighter objects will have been added to the ***AwaitingShipQueue*** and are standing by for the ***RoRoObj*** to request the first lighter to fill a vacant spot.

3. Replication

Replication is controlled by the replication manager object or ***RepMngrObj***. A method in this object asks the user how many replications are desired and if seeds for the random number generators are to be input by the user. At the end of each replication the desired statistics are computed and all of the lighters in the scenario are added to the ***AwaitingShipQueue*** in preparation for the next replication.

4. Output

Output is created in the Statistics object or ***StatsObj***. A method in this object is called after the last load from the ship has been transferred to the beach, and all of the lighters are returned to the ***AwaitingShipQueue***. This method calculates the desired statistics and outputs them to four ASCII files. Appendix E contains examples of program output files.

IV. ANALYTICAL PROCEDURES

A. BACKGROUND

In order to ensure that the output from the simulation model is reliable and correct, two steps must be accomplis ed. First, the model must be verified to ensure that it is mathematically correct. This essentially involves the checking and double checking of any equations, or input distributions used to determine event times in the model. The second step is validation of the model which, determines if the output is realistic and if the output is in fact what is required. In other words, does the simulation model provide reliable output data of the desired parameters. This two step process is the first phase in the analysis of the RO/RO Offload model.

The second phase in the analysis is to demonstrate the possible applications of the model. This is done by selecting a set of scenarios, running them through the model, and analyzing the output.

The following sections document the verification, validation, and demonstration of the RO/RO Offload simulation

model subject to the constraining assumptions listed in Chapter III.

B. LIGHTER CYCLE EVENT TIMES

In determining the times for each of the lighter cycle events the primary objective is to obtain the most current data available. Several sources were available that provided mean times and distributions for each of the events in the lighter cycle, but no one set of data provided both the accuracy and latitude that was required for the model. The final set used in the model is a mixed set derived from all of the sources available, coupled with some common sense decisions, to provide the best event times for this simulation model.

There are two capabilities in the RO/RO Offload model that required some flexibility in determining the event time distributions. The first is the capability to differentiate between self-sustaining (SSR) and non-self-sustaining (NSSR) RO/RO's. The second feature is that ships can be given the capability of LO/LO operations. Two additional area's that required flexibility are transit times, and refueling times.

Because the model allows the user to input the distances between the ships, beaches, and refueling area's, the transit times between area's are calculated based solely on speed and distance. This allows the user to place these area's at any distance relative to each other. Refueling times are also deterministic, based on the amount of fuel to be pumped, and the pump rate. The following table lists the times, distributions and source for each event.

TABLE 3. LIGHTER CYCLE EVENT INPUT DATA

LIGHTER CYCLE EVENT		CAUSEWAY FERRY			LCU		
		PROB DIST	MEAN/RNG	STD	PROB DIST	MEAN/RNG	STD
APPROACH & MOOR	SSR	NORMAL	10.5 (3)	3.2	NORMAL	14.3 (3)	2.2
	NSSR	NORMAL	8 (2)	3.2	NORMAL	14.3 (3)	2.2
OPERATIONAL DELAY 1		NORMAL	2 (1)	0.7	NORMAL	2 (1)	0.7
ONLOAD SHIP	SSR	NORMAL	16 (3)	3.9	NORMAL	15.9 (3)	3.9
	NSSR	NORMAL	25 (2)	3.9	NORMAL	18 (2)	3.9
OPERATIONAL DELAY 2		LOGNORMAL	1.2 (1)	1.2	NORMAL	1 (1)	0.3
CAST & CLEAR SHIP	SSR	NORMAL	5 (3)	1.3	UNIFORM	2-2.5 (3)	NA
	NSSR	NORMAL	4 (2)	1.3	UNIFORM	4-6.5 (2)	NA
TRANSIT TO BEACH		DETERMINISTIC			DETERMINISTIC		
APPROACH & MOOR		NORMAL	17 (1)	3.4	NORMAL	11 (1)	4.29
OPERATIONAL DELAY 3		LOGNORMAL	1 (1)	0.9	UNIFORM	1-3 (1)	NA
OFFLOAD AT BEACH		NORMAL	10 (1)	3.4	NORMAL	3 (1)	0.8
OPERATIONAL DELAY 4		UNIFORM	0-1.5 (1)	NA	NORMAL	1 (1)	0.3
CAST & CLEAR BEACH		NORMAL	9.9 (1)	1.8	UNIFORM	1.8-3 (1)	NA
TRANSIT TO SHIP		DETERMINISTIC			DETERMINISTIC		

NOTES: a. All values are in minutes.
b. Onload and Offload times are in minutes per vehicle.
SOURCES: (1) Ref. 8: p49. Note: All STD's and distributions come from this reference.
(2) Ref. 8: p34. Note: Allows differentiation between SSR and NSSR ships.
(3) Ref. 9: p6-18 Note: Most current data available for these events.

TABLE 4. LIGHTER CYCLE EVENT INPUT DATA

LIGHTER CYCLE EVENT	CAUSEWAY FERRY			LCU		
	PROB DIST	MEAN/RNG	STD	PROB DIST	MEAN/RNG	STD
TRANSIT TO REFUEL AREA	DETERMINISTIC			DETERMINISTIC		
APPROACH & MOOR	NORMAL	17 (1)	3.4	NORMAL	11 (1)	4.29
OPERATIONAL DELAY 3	LOGNORMAL	1 (1)	0.9	UNIFORM	1-3 (1)	NA
REFUEL	DETERMINISTIC			DETERMINISTIC		
CAST & CLEAR REFUEL AREA	NORMAL	9.9 (1)	1.8	UNIFORM	1.8-3 (1)	NA
TRANSIT TO SHIP	DETERMINISTIC			DETERMINISTIC		

NOTES: a. All values are in minutes.
b. Refuel Area is located at a beach or shore site, therefore it is treated as a beach for simulation event times.
SOURCES: (1) Ref. 8: p49. Note: All STD's and distributions come from this reference.

Two additional notes on event times that are not reflected in the tables. The first is that an LSV is treated as an LCU with regard to event times. Since an LSV moors to the RRDF at the LCU spot, and is similar in most ways to an LCU, this is not regarded as a risky assumption. This assumption was further required because no data was available for the LSV event times. The second thing to note is the offload time at the LO/LO spot if the RO/RO being simulated is so configured. The event time for a LO/LO offload is normally distributed with a mean of 10.25 minutes per vehicle and a standard deviation of 5.75 minutes per vehicle. These values are a composite based on data obtained in *CRM*

91-3, [Ref. 10], and data from *Joint Pub 4-01.6,* [Ref. 1]. The standard deviation for the LO/LO offload time as well as three other event times is actually too high for the normal or lognormal distributions. In a simulation run, a small percentage of the numbers drawn from the random number streams in these cases would be negative. This would cause a fatal error in the program, thus, these numbers are converted to positives. As a result, the distributions actually used in the model are truncated and slightly skewed to the right of those described.

C. ANALYSIS

The analysis of model output is divided into two parts. The stated goal of this thesis is to provide a valid simulation model from which reliable estimates of throughput can be derived for the instream offload of a RO/RO. The first objective is to validate the simulation model, thus ensuring that it does in fact provide reliable throughput estimates. Step two is to demonstrate the use of the model using four scenarios and evaluating the output.

1. Validation Scenario

Validation of the RO/RO Offload model was accomplished by taking a real world offload scenario for which all of the parameters required by the model were known. These parameters were loaded into the model via the input files. The RO/RO Offload model was then run for 300 replications and the output was compared with the existing empirical data. The operation selected for validation was actually conducted during Ocean Venture '93. The following tables list the input values for the model. For examples of input files in the correct format, see Appendix D.

TABLE 5. VALIDATION SCENARIO INPUT DATA

SHIP DATA		REFUEL AREA DATA	
NAME	BELLATRIX	NAME	FUEL DEPOT
TYPE	SSR	# OF SPOTS	2
DIST TO BCH	6 nm	DIST TO SHIP	6 nm
# OF SPOTS	3	PUMP RATE	3500 GPH
SPOT TYPE	LCU, CWF, LOLO		
# VHCLS LOLO	60		
# VHCLS RRDF	0		
# VHCLS ANY SPOT	834		

NOTES: a. # VHCLS LOLO indicates the number of vehicles in the load that can be lifted from the LOLO spot only. The Bellatrix, an SL-7 class, has a partitioned load and 60 vehicles must be offloaded by LOLO.
b. Data provided by Joint Test Directorate [Ref. 11].

TABLE 6. VALIDATION SCENARIO INPUT DATA

LIGHTER DATA				
LIGHTER ID	ALFA - ECHO	FOXTROT - GOLF	HOTEL	INDIA - LIMA
TYPE	LCU 2000	LCU 1610	LSV	3 X 1 CWF
# IN SCENARIO	5	2	1	4
SPOT REQUIRED	LCU	LCU	LCU	CWF
MAX SPEED	12 kts	12 kts	11.6 kts	6 kts
FULL LOAD SPEED	11 kts	11 kts	10 kts	3 kts
VHCLE CAPACITY	11	4	25	16
FUEL CAPACITY	92000.0 GAL	3290.0 GAL	165000.0 GAL	275.0 GAL
FUEL BURN RATE	41.6 GPH	36.0 GPH	145.8 GPH	20.8 GPH
MIN FUEL %	0.25	0.25	0.25	0.25

NOTES: a. Fuel capacities, Burn rates, and vehicle capacities for the LCU 2000, LSV, and 3 x 1 CWF provided by Joint Test Directorate [Ref. 11].

TABLE 7. VALIDATION SCENARIO INPUT DATA

BEACH DATA				
NAME	SOUTH	ARMY	NAVY	ADMIN
TYPE	BARE BEACH	CAUSEWAY PIER	CAUSEWAY PIER	CAUSEWAY PIER
# OF SPOTS	2	2	1	1
DIST TO SHIP	6 nm	6 nm	6 nm	6 nm
DIST TO FUEL	6 nm	6 nm	6 nm	6 nm

NOTES: a. Data provided by Joint Test Directorate [Ref. 11].

2. Validation Analysis

The approach employed in this thesis for model validation is the *basic inspection approach* [Ref. 12: p. 316]. This procedure involves the comparison of one or more statistics from the model with those from real world

observations. For this model the total offload time for the ship in the given scenario will serve as the statistic for comparison. Although the RO/RO Offload model collects data for the idle time at each of the ship and beach spots in the scenario, as well as the time spent in the ***AwaitingShipQueue*** and ***AwaitingBeachQueue***, there currently is no empirical data with which to compare it.

The RO/RO Offload model is a terminating simulation [Ref. 12:p. 529]. That is to say that each replication of the simulation runs until a terminating event occurs, namely, the complete offload of the ship in the given scenario. Since different runs use an independent sequence of random numbers for the individual event times, this implies that the output realizations from the different runs are independent as well. Specifically, total offload time for the ship in our scenario. Calculating the mean $\bar{X} = E(X)$, where X is the independent total offload time for a single simulation replication, serves as a reliable point estimate for the comparison. If $X_1, X_2, \ldots, X_n$ are the independent, realizations for offload times of n replications, then the mean is simply: $\bar{X} = (\Sigma_{i=1}^{n} X_i)/n$

As noted previously, the validation run of the model consisted of 300 replications. The output is in ASCII format, examples of which can be found in Appendix E. The output was loaded into SPSS [Ref. 13], and some basic graphical analysis was conducted. Output times from the model are expressed in minutes, and the real world observations are generally expressed in hours or days, therefore, some simple conversion is required. The results for the validation are presented below in Figure 4.

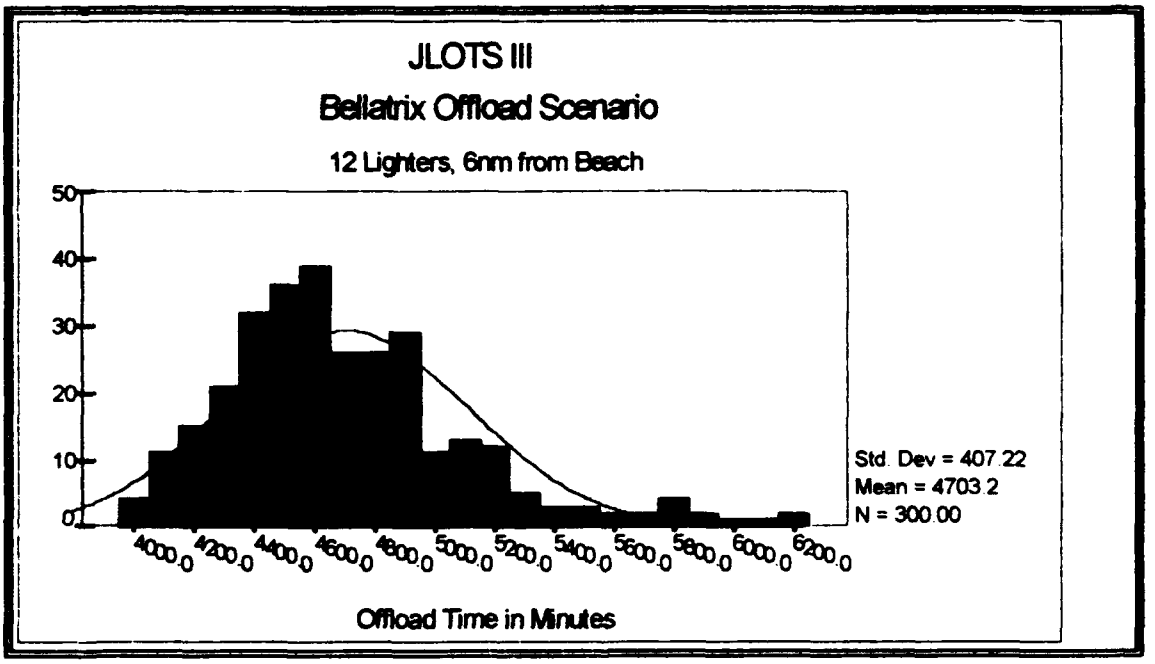

Figure 4: Histogram of Bellatrix Offload Scenario Compared to Normal Distribution.

We expect our offload time data to be slightly skewed to the right since offload time is bounded by zero on

the left. This is confirmed by Figure 4. Nevertheless, we rely on the central limit theorem and the robustness of procedures based on the t-distribution to build a 99% confidence interval for the mean offload time using the 300 simulated observations.

$$\bar{X} \pm z_{\alpha/2}(s/\sqrt{n}) = 4703.2 \pm 2.576(407.22/\sqrt{300}) = [4642.63, 4763.76]$$

This interval expressed in hours is [77.37, 79.39]. The empirical mean, 79.2, clearly falls within this confidence interval which lends credence to our claim that the RoRo Offload simulation model provides reliable estimates for throughput.

3. Demonstration Scenarios

In order to fully exercise the simulation model, a design space consisting of four scenarios was selected. There are many factors which could be varied to determine the effect on total offload time, but the two most obvious variables are the number of lighters, and the distance of the ship from the beach. The objective is to stress the model, thus, the values for these variables could be considered extreme cases that are not likely to be

replicated in the real world. The four scenarios are, a large number of lighters at both long and short distances, and a small number of lighters at both long and short distances. In choosing the number of lighters, some experimental runs were conducted to ensure that the queues in the model were behaving as desired. Aside from the number of lighters, and the distance between ship and beach, all other variables were held constant. The data used in the validation scenario listed in tables 5, 6, and 7 were used with the exception of distances, and lighters. The number of beaches was reduced to two, the *south* beach and the *admin* pier. The number of beaches was reduced to help obtain the desired behavior in the queues. Figure 5 depicts the design space for the demonstration.

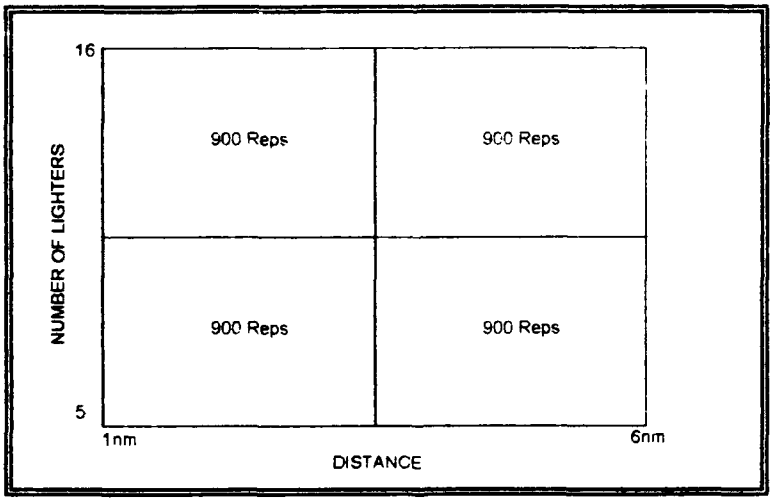

Figure 5: Demonstration Design Space

4. Demonstration Results

Graphical techniques were employed to analyze the data from the four demonstration scenarios. The model was run for n = 900 replications for each case. Figure 6 depicts the results from the four runs.

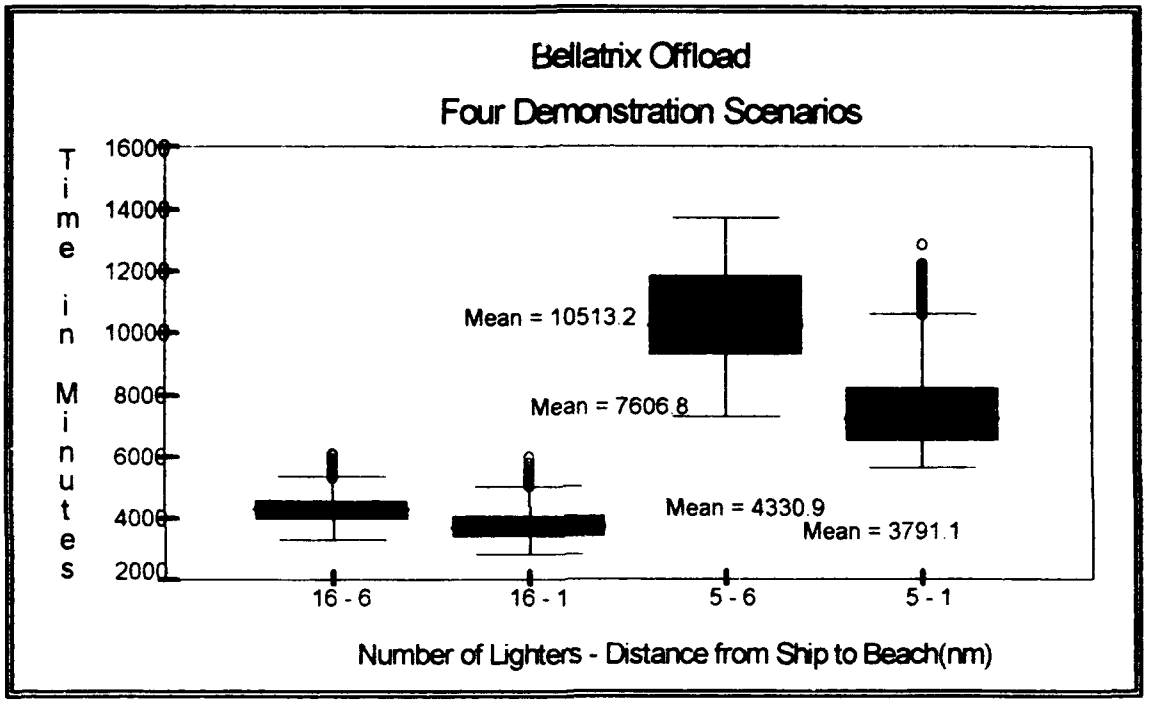

Figure 6: Boxplots of Results from Demonstration Scenarios

Each of the four boxplots above depicts the results from one of the four scenarios. The first observation to be made is that the two 16 lighter scenarios are relatively close in their final values. In fact, there is only approximately 9 hours difference between these two mean offload times. This difference is likely due to the

difference in distance between the two scenarios. We let X_1 and X_2 be the observed offload times for a replication of the 16-6 and 16-1 scenarios respectively, and then ran a paired t-test to compare the two scenarios where:

$$H_o : \bar{X}_{i,1} - \bar{X}_{i,2} = D_i = 0, \text{ and}$$

$$t = \frac{\bar{D}}{S_D \sqrt{N}},$$

where $\bar{D} = \Sigma_{i=1}^{n} D_i/n$ and S_D is the standard deviation of the differences of paired observations. The results of the paired t-test follow:

TABLE 8. PAIRED T-TEST FOR 16-6 AND 16-1 SCENARIOS

Mean	4,330.86	3,791.11
Standard Deviation	429.91	503.49
SE of Mean	14.33	16.78
Number of Pairs	900	
Correlation	0.02	
2-tail Significance	0.54	
Paired Differences		
Mean	539.75	
Standard Deviation	655.32	
SE of Mean	21.84	
99% Confidence	(483.351 596.141)	
t-value	24.71	
df	899	
2-tail Significance	0	

A two-tail significance of zero for the paired differences indicates that the probability of observing a difference this large or larger when the two means are the same is essentially zero. Additionally, zero does not fall within the 99% confidence interval for the mean of the paired differences. The conclusion is that H_o must be rejected and the difference between the two means is statistically significant. Although the difference is statistically significant, it would be deemed operationally insignificant in the course of an entire operation. This difference is relatively small due to the buffering effect of the queues. The times spent in queue by lighter type and the idle spot times are listed below.

TABLE 9. MEAN TIME SPENT IN QUEUE BY LIGHTER TYPE

Scenario	Awaiting Ship Queue Data			Awaiting Beach Queue Data		
	LCU Ship	LSV Ship	CWF Ship	LCU Beach	LSV Beach	CWF Beach
16-6	1,066.5	691.9	725.3	144.3	69.5	39.5
16-1	1,198.3	761.3	840.4	418.9	187.7	166.6
5-6	584.9	0	0	0.4	0	0
5-1	1,047.7	0	0	3.6	0	1.2

NOTES: a. All times in minutes

TABLE 10. MEAN TIME SHIP AND BEACH SPOTS WERE IDLE

Scenario	Ship Idle Spot Time			Beach Idle Spot Time	
	LCU Spot	CWF Spot	LoLo Spot	South Beach	Admin Pier
16-6	1,788.8	1,508.4	1,733.4	1,643.33	2,092.1
16-1	1,615.9	1,210.2	1,646.1	1,198.6	1,518.8
5-6	2,764.7	6,239.7	2,883.4	6,051.6	6,950.6
5-1	1,724.6	3,679.2	1,782.2	4,074.1	5,207.4

NOTES: a. All times in minutes

While the interpretation of the total offload time is straightforward, care must be taken when interpreting these times. First, note that the mean time spent in queue is bounded by the total offload time, and should be normalized to the percent of the total offload time spent in queue. Second, note that the idle times have some non-zero lower bound that is determined by the number of loads carried by each lighter type.

With so many lighters in the 16 lighter scenarios, the queues rarely are without the desired lighter type for a newly available spot, thus, these offload cycles have a minimum idle time at both the ship and at the beach. Likewise, the time spent in queue is relatively high because of the large number of lighters. When the distance is decreased from six to one mile, the time spent in queue

increases and the idle spot times decrease, as would be expected.

In contrast, the two scenarios with five lighters each had little or no buffering from the queues, thus, the idle spot times increase significantly. The mix of lighters must be explained to understand fully what is happening in these scenarios. The lighter mix for the five lighter scenarios consists of four LCU's, and one CWF. Since there is no LSV in the scenario, LSV queue times are zero. Because there is only one CWF in the scenario, the time spent in queue for the CWF is zero, and it's associated idle spot time is high. The CWF is essentially running back and forth directly between ship and beach. For the LCU's, they are spending more time in the queues because there are more of them in the mix and they are competing for the same spots. This undoubtedly forces some into the queue. The queue times are significantly lower than in the 16 lighter scenarios, and the idle times are also higher, as would be expected. Again, as the distance decreases, the queue times increase and the idle times decrease.

What can be concluded about the five lighter scenarios is that scarcity of resources is driving up the total offload time. The increase in distance form one to six miles makes the lighter resources even more scarce, thus, a higher offload time.

It must be noted that without empirical data to compare with, it is difficult to say what the queue times and the idle times should be. There is a significant amount of variance that results from a mix of lighters with different load capacities. What has been found in empirical observations is that using a mix of lighters with similar capacities smoothes the process. The model results support this finding. In the five lighter scenario we used four LCU's. If an LSV with a 25 vehicle capacity is substituted for on LCU in the five lighter mix, the awaiting ship queue time can be driven up even further. With fewer lighters in a scenario the effect is more pronounced.

What has been demonstrated here is that the RO/RO Offload model can be useful in determining throughput time for a given scenario. In real world operations the majority of variables will be fixed. The decision maker will have a

smaller set of variables that can be adjusted to improve the performance in an offload. For example, the distance of the ship from the beach is probably a function of anchorage locations, thus limiting the options. For a given distance and ship type, the LOTS planner can use the RO/RO Offload simulation model to help find a better mix of lighters to work within the fixed parameters.

V. CONCLUSIONS

A. CONCLUSIONS

The purpose of this thesis was to provide a valid model with which reliable estimates of RO/RO throughput could be obtained. The RO/RO Offload model is valid for this purpose within the scope of the limiting assumptions. The model can be used effectively not only to estimate throughput, but to possibly improve throughput rates by providing LOTS planners the opportunity to model an offload and adjust the variables long before the real world equipment is on station.

The four demonstration scenarios serve purely as an illustration of how the model may be used. Planners are certainly not limited to altering only these variables. In theory, the model could have been run hundreds of times, evaluating every possible combination of variables. The run time for the model is so short, however, that it may be used on demand to provide planning factors tailored to a specific scenario.

B. RECOMMENDATIONS

The RO/RO Offload model also has the ability to collect data on the mean time spent in each of the queues by lighter type as well as the idle time for ship and beach spots. It is hoped that by collecting and analyzing this data that some insight could be gained as to how best to alter the number and mix of lighters for a scenario. Empirical data was not available to validate these features, however, they were demonstrated in this thesis. We recommend that in future operations, data be collected so that the model can be further validated.

Regarding future work, there are several features that could be added to either add fidelity to the model or make it more user friendly. They are:

1. Add graphics to the model.
2. Modify the model to allow the offload of container ships. The modular structure of object-oriented programming lends itself to these seemingly broad modifications. The objects in the RO/RO Offload model are generic enough to make this a relatively simple task.
3. Incorporate weather and sea state into the model.
4. Incorporate equipment failures into the model.

LIST OF REFERENCES

1. Joint Publication 4-01.6, The Joint Staff, *Joint Tactics, Techniques, and Procedures for Joint Logistics Over The Shore*, August 1991.

2. Telephone conversation between Captain Steve Christy, JLOTS III Test Directorate, and the author, 17 May 1993.

3. Telephone conversation between LtCol David Miller, J-4 Mobility, The Joint Staff, and the author, 18 May 1993.

4. JLOTS II Test Directorate, *JLOTS II Throughput Test*, Naval Amphibious Base, Little Creek, Norfolk, VA, 1985.

5. Department of the Army, *LOGEX 88 Joint Logistics Over The Shore (JLOTS) Exercise Executive Summary*, U.S. Army Transportation Center, Fort Eustis, VA, 1989.

6. CACI Products Company, *MODSIM II The Language for Object-Oriented Programming, Reference Manual*, La Jolla, CA, January 1992.

7. Bailey, Michael, "Object-Oriented Simulation Pictures (OOS-PIC) For Designing and Testing," Naval Postgraduate School, Monterey, CA, 1 February 1993.

8. JLOTS II Test Directorate, *JLOTS II Roll On/Roll Off Ship Operations*, Naval Amphibious Base, Little Creek, Norfolk, VA, 19 March 1984.

9. JLOTS III Te Directorate, *JLOTS III Display Determination 91 Test Report*, Naval Amphibious Base, Little Creek, Norfolk, VA, 18 November 1992.

10. Center For Naval Analysis, *Theoretical Distributions of Maritime Prepositioned Force Barge Cycle Component Times*, CRM 91-3, by William A. D. Wallace, February 1991.

11. Telephone conversation between Captain Steve Christy, JLOTS III Test Directorate, and the author, 17 August 1993.

12. Law, A. M., and Kelton, W. D., *Simulation Modeling and Analysis*, 2d ed., McGraw-Hill, 1991.

13. Norusis, M. J., *SPSS For Windows Base System Users Guide, Release 5.0*, SPSS, Chicago, IL, 1992.

BIBLIOGRAPHY

Bailey, Michael, "RecIOHandleObj: Object Input Made Easy," Naval Postgraduate School, Monterey, CA, 1 February 1993.

Center For Naval Analysis, CRM 89-339, *MPF Exercise Summary*, by John F. Nance and William A. D. Wallace, February 1990.

Center For Naval Analysis, CRM 91-101, *The Maritime Prepositioned Force Offload Model*, Vol. I, by C. A. Cowie, R. W. Reichard and W. A. D. Wallace, April 1991.

David Taylor Research Center, *Evaluation of JLOTS Lessons Learned in Solid Shield 89*, by The Mobile Support Systems Office (Code 1250), September 1989.

JLOTS III Test Directorate, *JLOTS III Display Determination 91 Test Report*, Naval Amphibious Base, Little Creek, Norfolk, VA, 18 November 1992.

JLOTS II Test Directorate, *JLOTS II Throughput Test*, Naval Amphibious Base, Little Creek, Norfolk, VA, 1985.

Law, A. M., and Kelton, W. D., *Simulation Modeling and Analysis*, 2d ed., McGraw-Hill, 1991.

Mendenhall, W., Wackerly, D. D., and Scheaffer, R. L., *Mathematical Statistics with Applications*, 4th ed., PWS-KENT, 1990.

Naval Civil Engineering Laboratory, Report 1717, *Productivity Analysis of Powered Causeway Sections for ContainerShip Offloading*, by R. E. Bergman, December 1984.

Shaw, S. E., *An Object-Oriented Ship-To-Shore Movement Analysis Model (CUTTER)*, Master's Thesis, Naval Postgraduate School, Monterey, CA, September 1992.

Speight, J. A., *A Management Decision Model For Logistics Over The Shore (LOTS) Operations Using Conditional Stochastic Network Techniques,* Master's Thesis, Massachusetts Institute of Technology, Cambridge, MA, January 1988.

Sumner, J. D., *An Analysis of the Maritime Prepositioning Ship (MPS) Instream Offload: A Decision Framework for the Marine Corps Commander*, Master's Thesis, Naval Postgraduate School, Monterey, CA, December 1991.

Zalewski, A. J., *A Ship-To-Shore Simulation*, Master's Thesis, Naval Postgraduate School, Monterey, CA, September 1991.

APPENDIX A LIST OF ACRONYMS

BLCP	Beach Lighterage Control Point
CSNP	Causeway Section Non-Powered
CSP	Causeway Section Powered
CWF	Causeway Ferry
DOD	Department of Defense
ELCAS	Elevated Causeway Pier
GPH	Gallons per Hour
JLOTS	Joint Logistics Over The Shore
LO/LO	Lift On/Lift Off
LOTS	Logistics Over The Shore
LCU	Type of Lighter, LCU 1466, LCU 1610, LCU 2000
LSV	Logistics Support Vessel
NSSR	Non-Self-Sustaining RO/RO
OPDS	Offshore Petroleum Discharge System
RO/RO	Roll On/Roll Off
RRDF	Roll On/Roll Off Discharge Facility
SLCP	Ship Lighterage Control Point
SSR	Self-Sustaining RO/RO
T-ACS	Crane Ship

APPENDIX B OBJECT-ORIENTED SIMULATION PICTURES

The following pages document the Transition action diagrams for the RO/RO Offload model. Transition action diagrams are a combination of old style flow charts, coupled with Object-Oriented simulation pictures, or OOS-Pics. Together they show the flow of control in the Lighter Cycle operation of the RO/RO Offload model, including the interactions between objects, and their methods and fields.

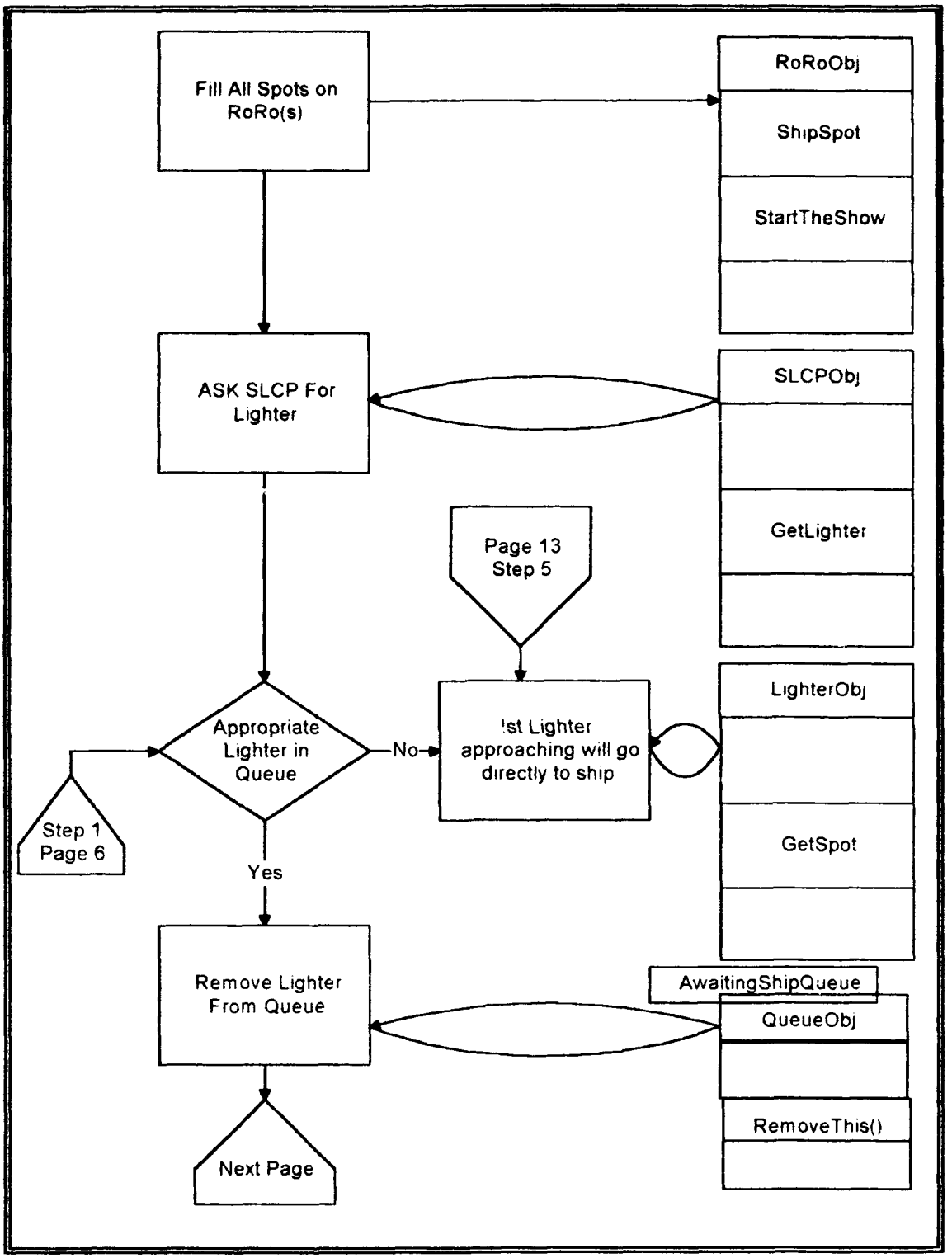

Figure 7: OOS-Pics Page 1.

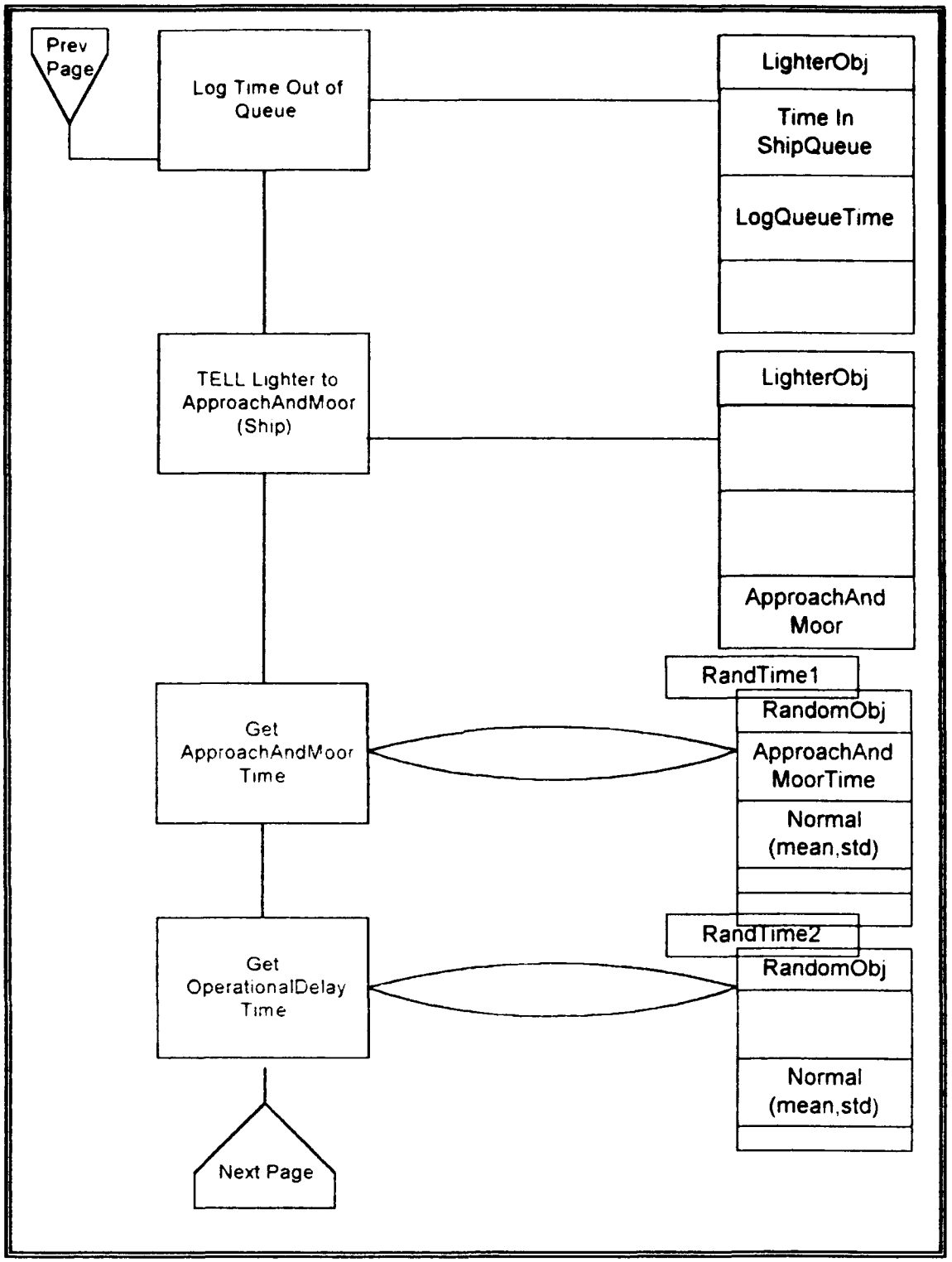

Figure 8: OOS-Pics Page 2.

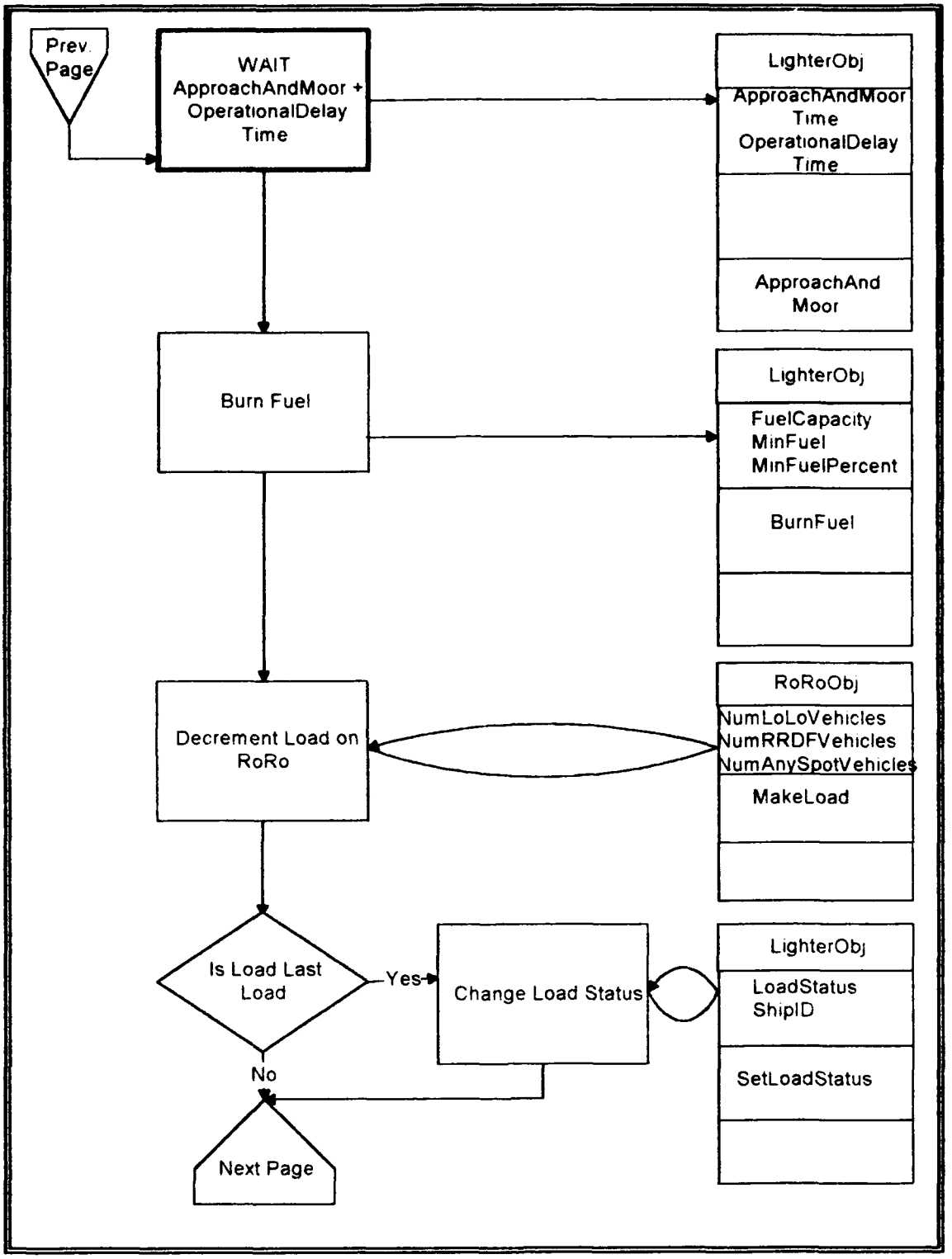

Figure 9: OOS-Pics Page 3.

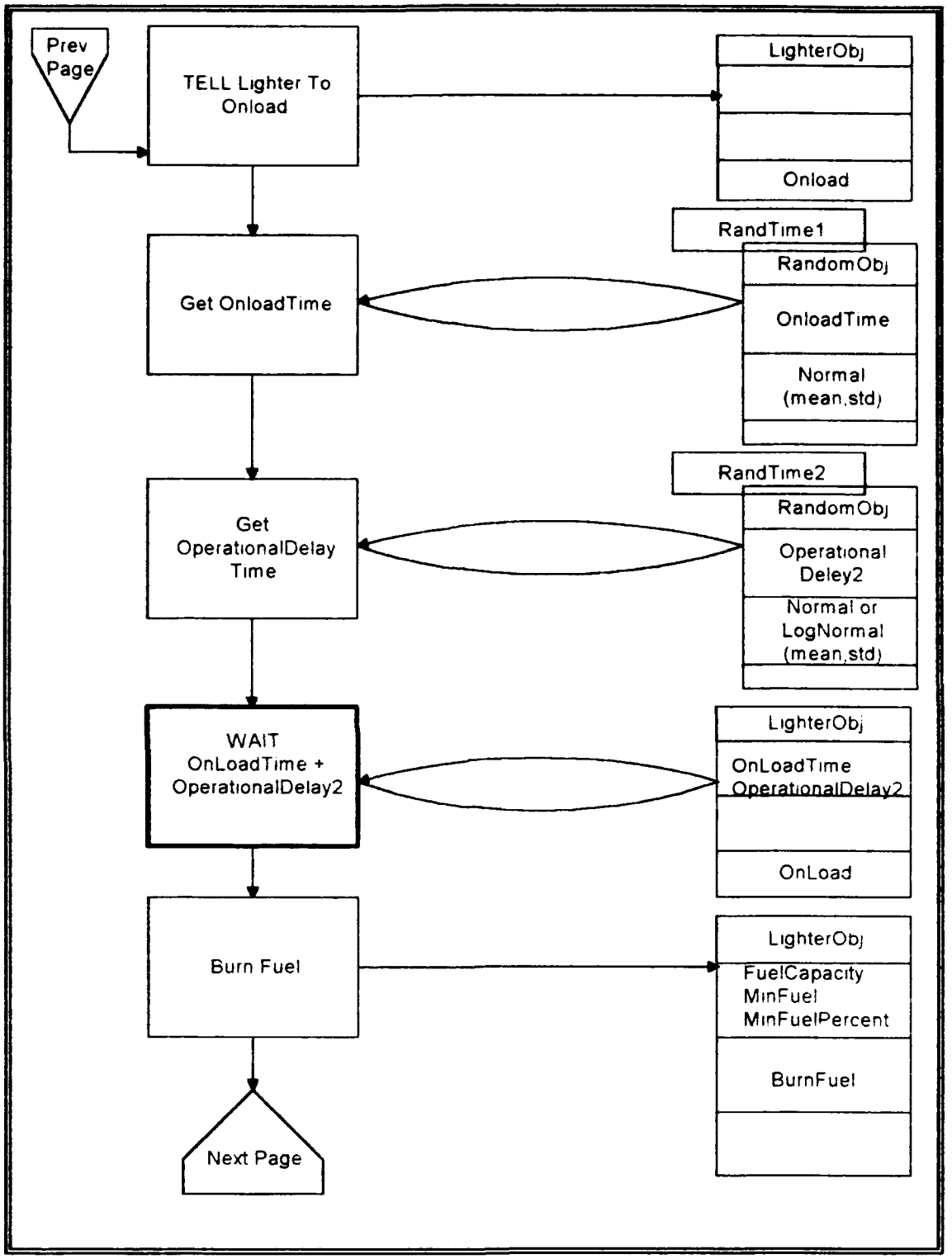

Figure 10: OOS-Pics Page 4.

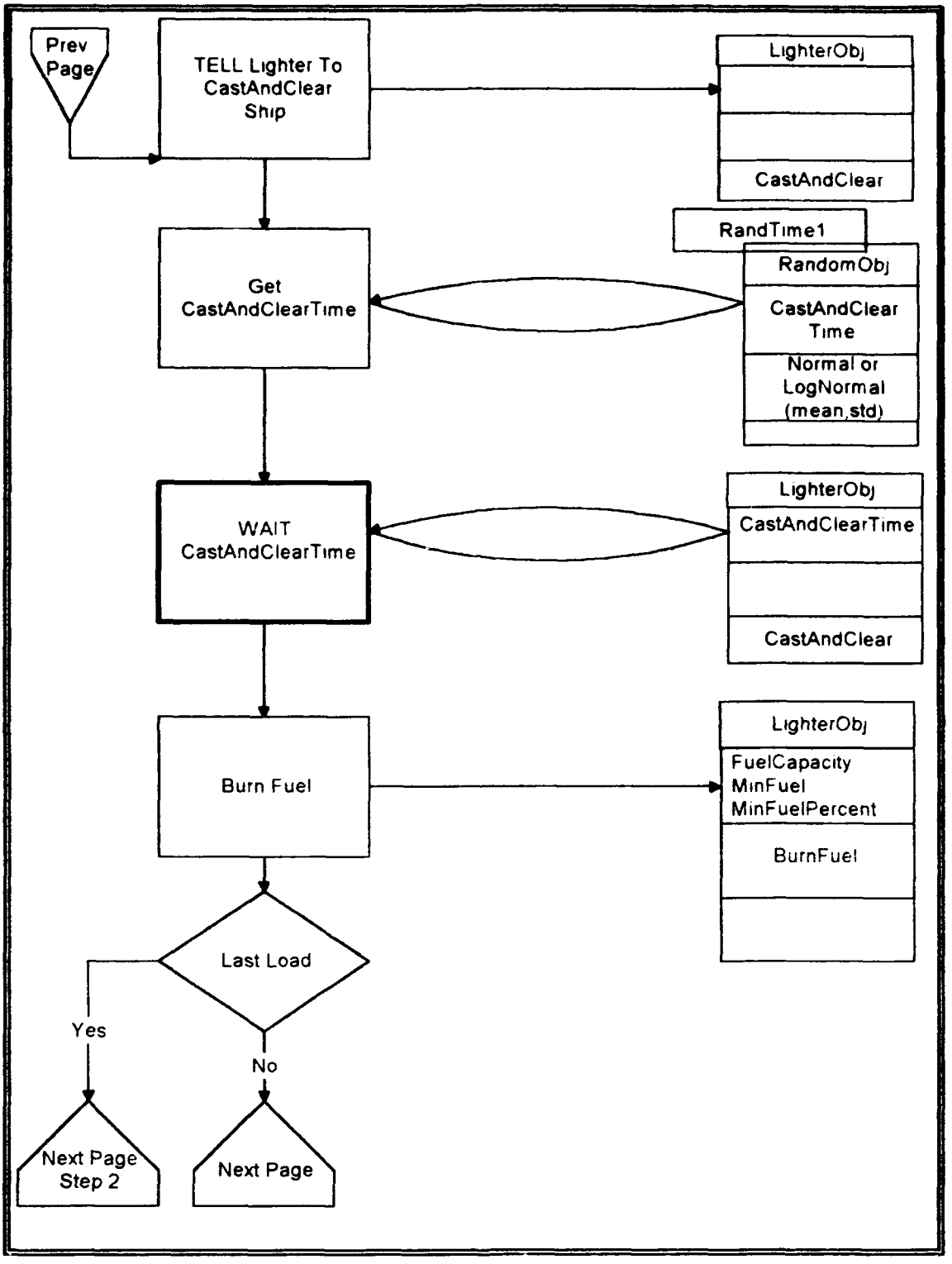

Figure 11: OOS-Pics Page 5.

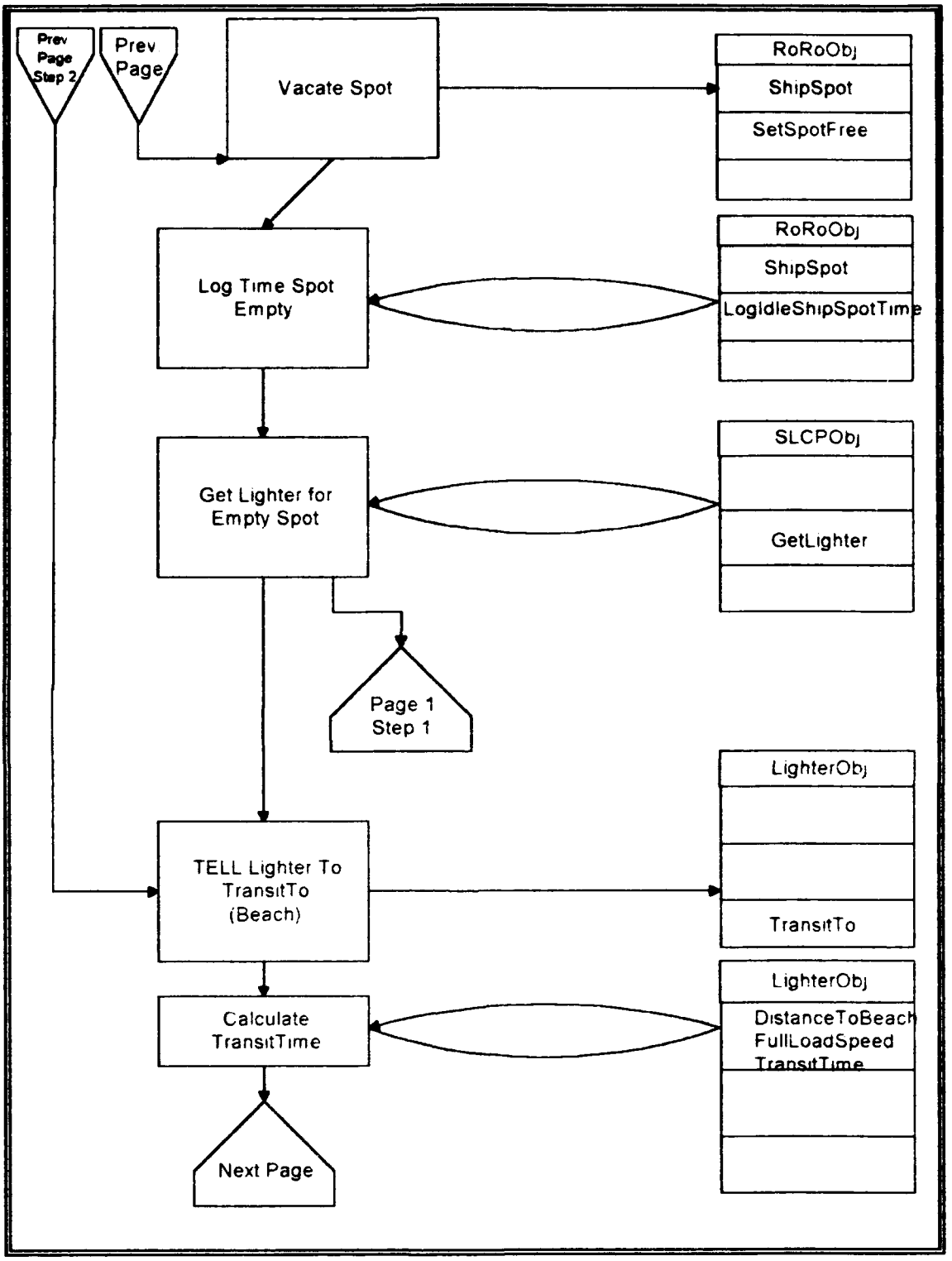

Figure 12: OOS-Pics Page 6.

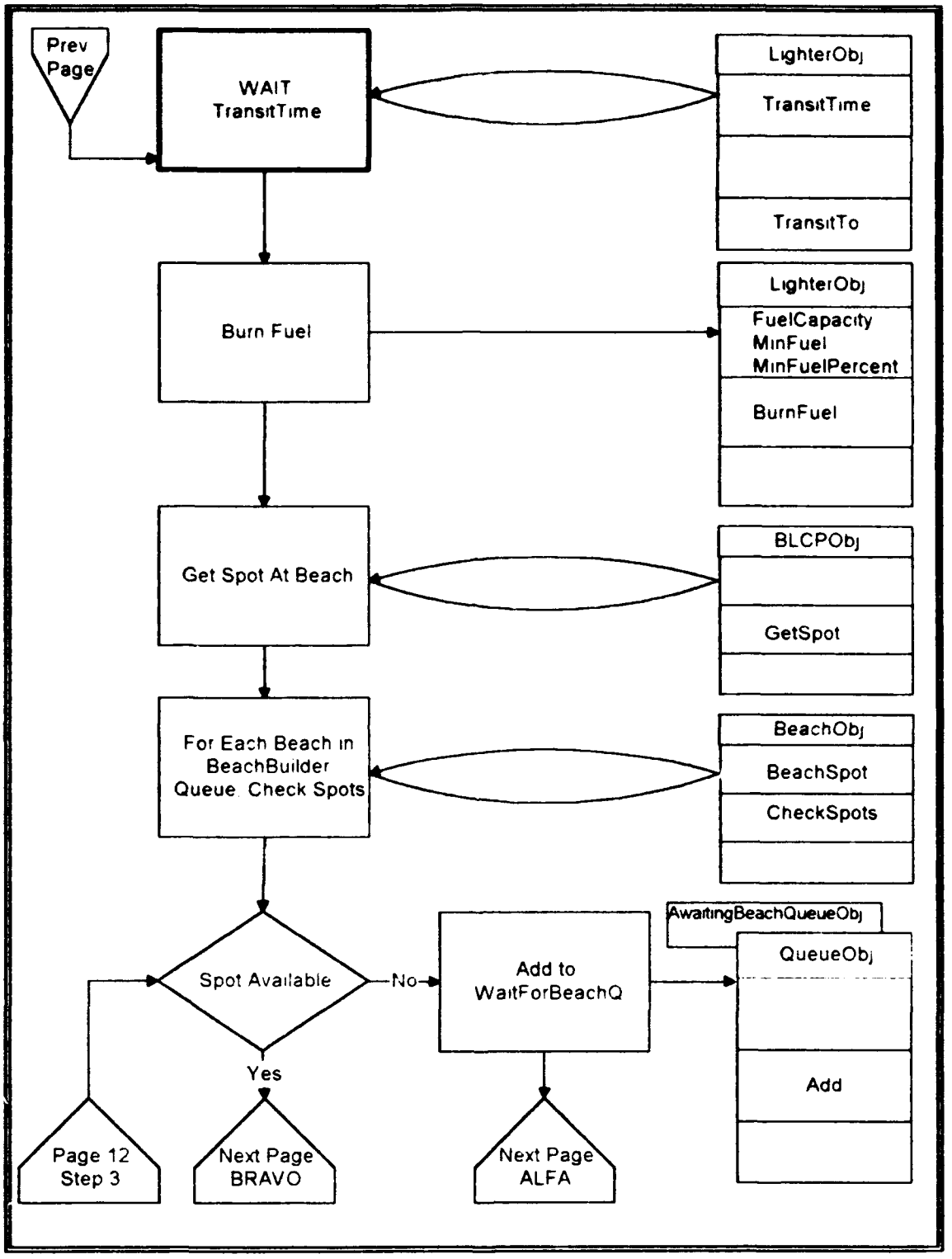

Figure 13: OOS-Pics Page 7.

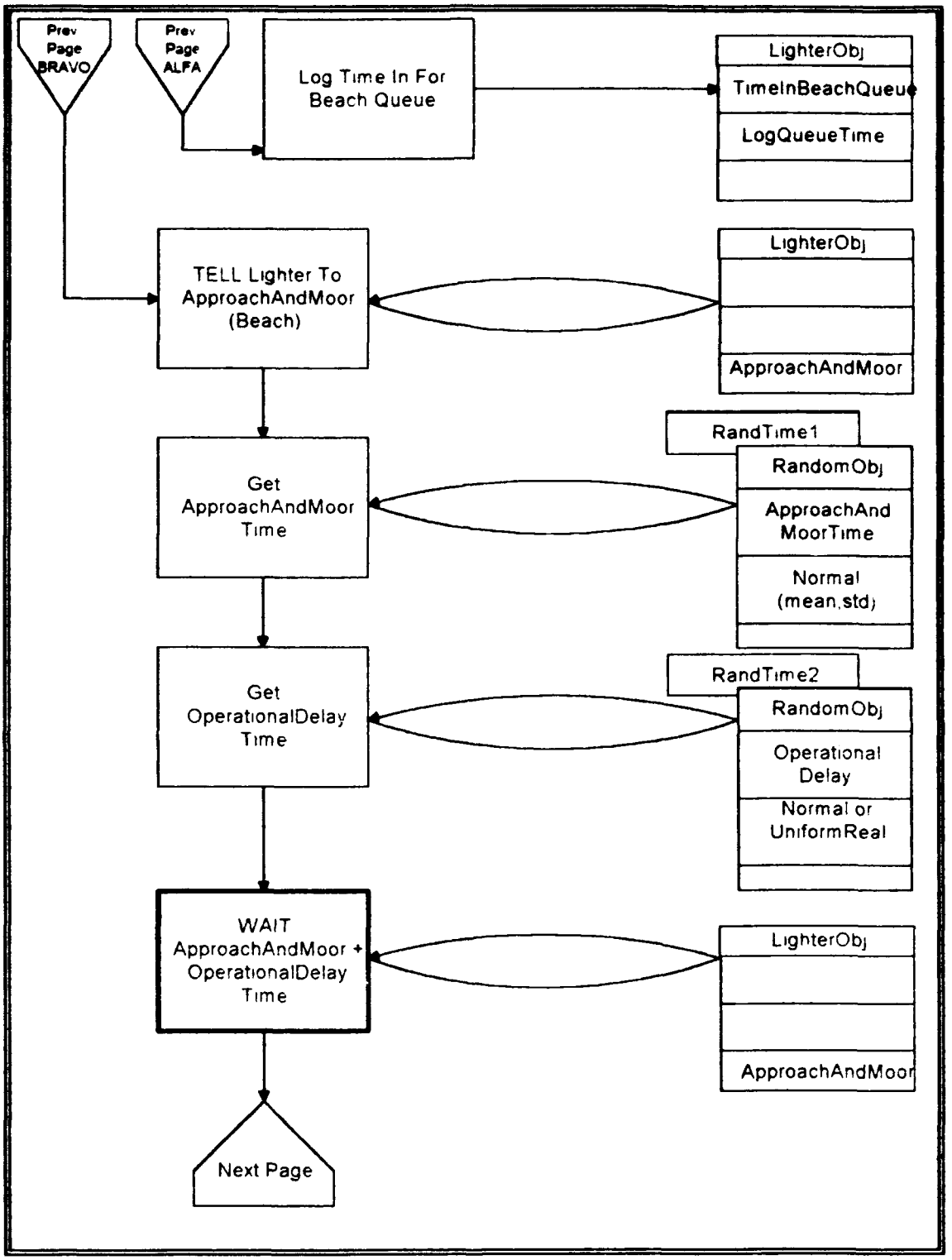

Figure 14: OOS-Pics Page 8.

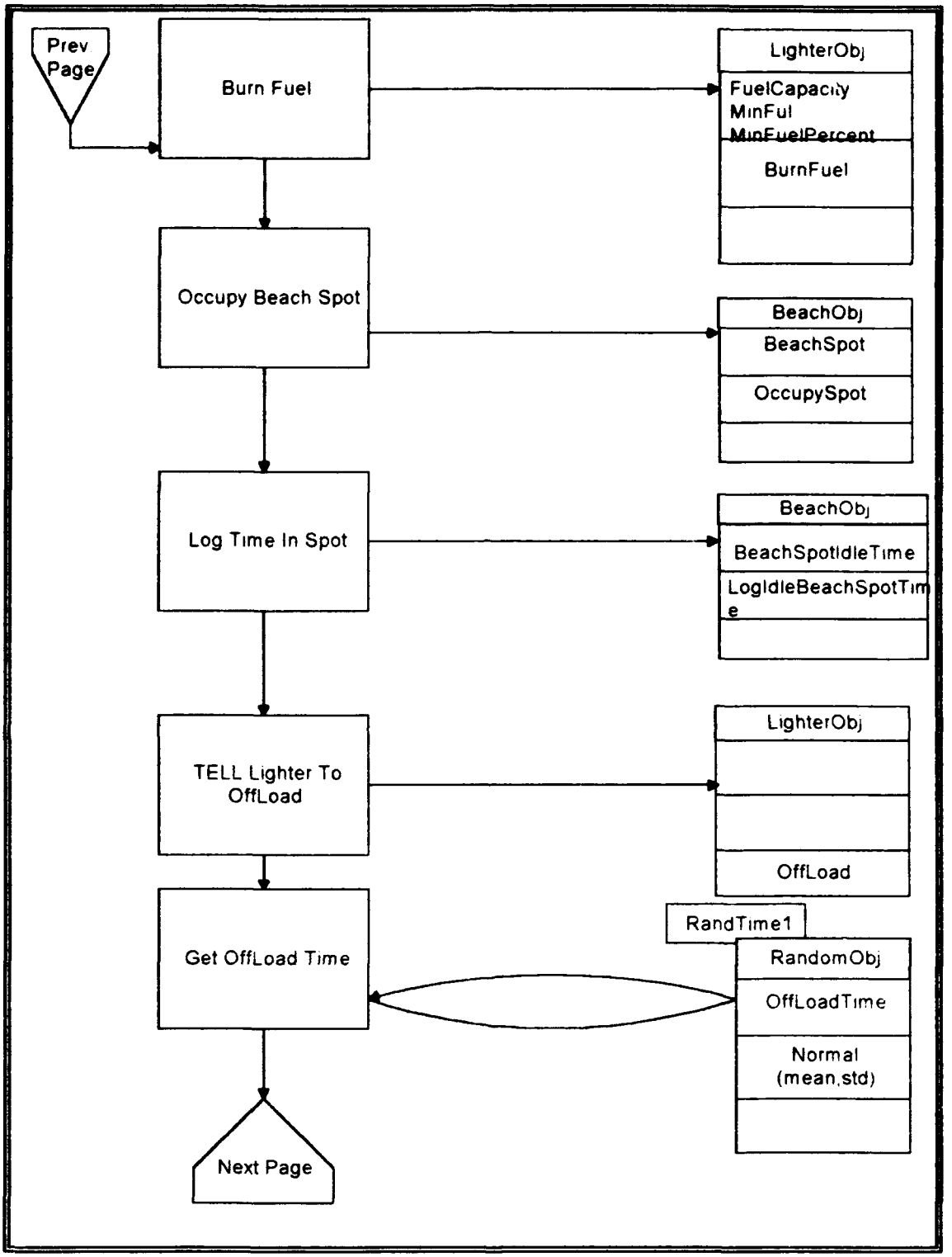

Figure 15: OOS-Pics Page 9.

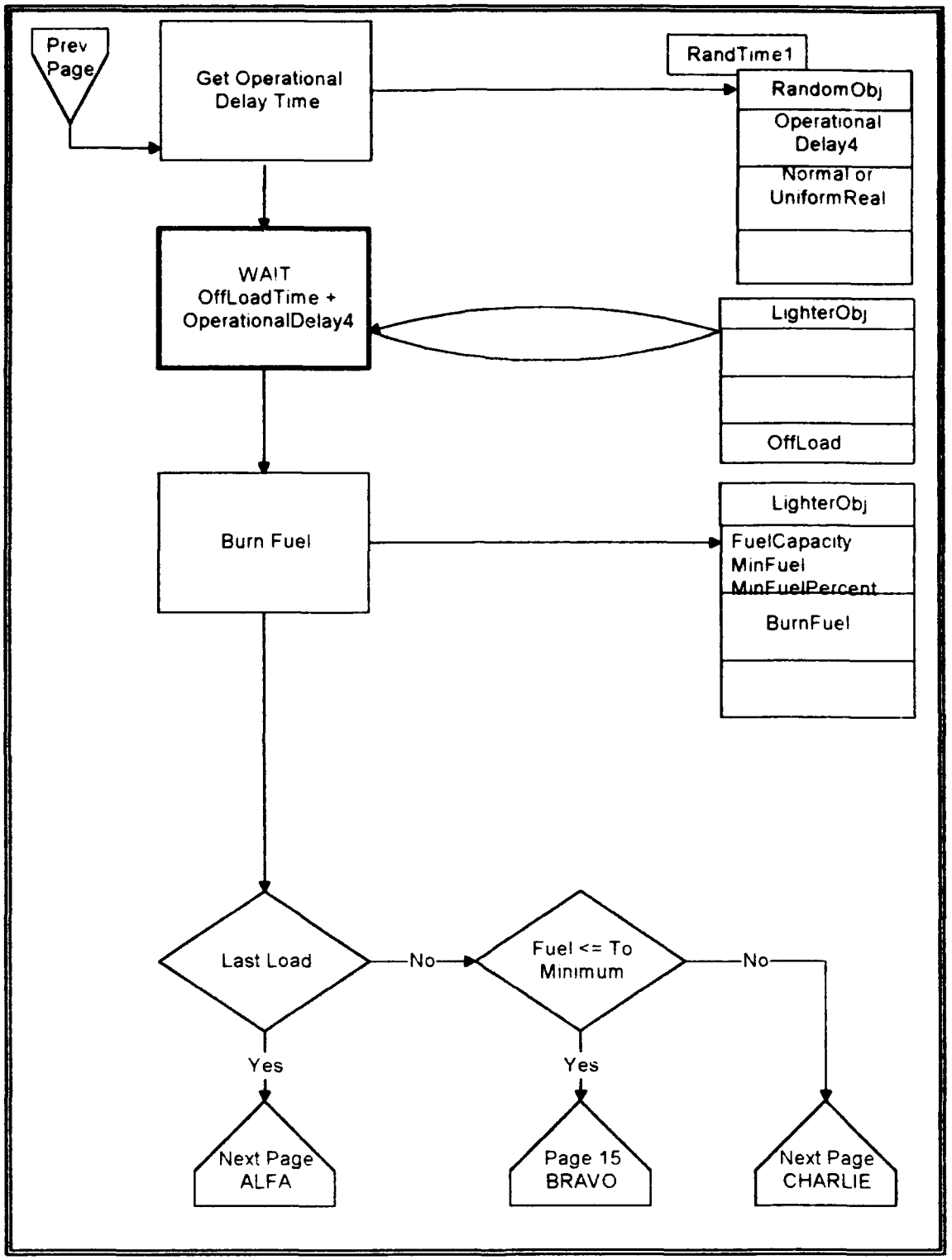

Figure 16: OOS-Pics Page 10.

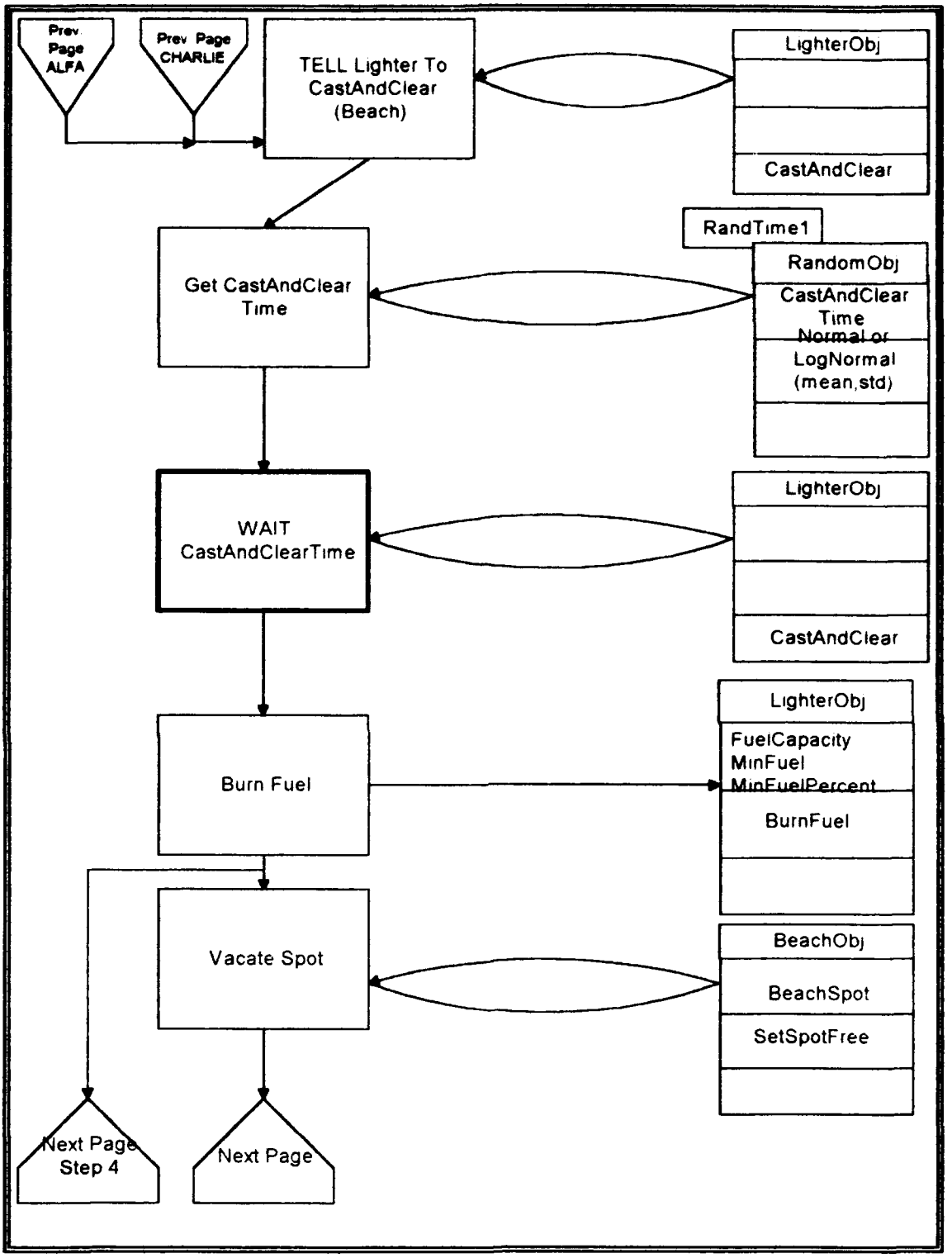

Figure 17: OOS-Pics Page 11.

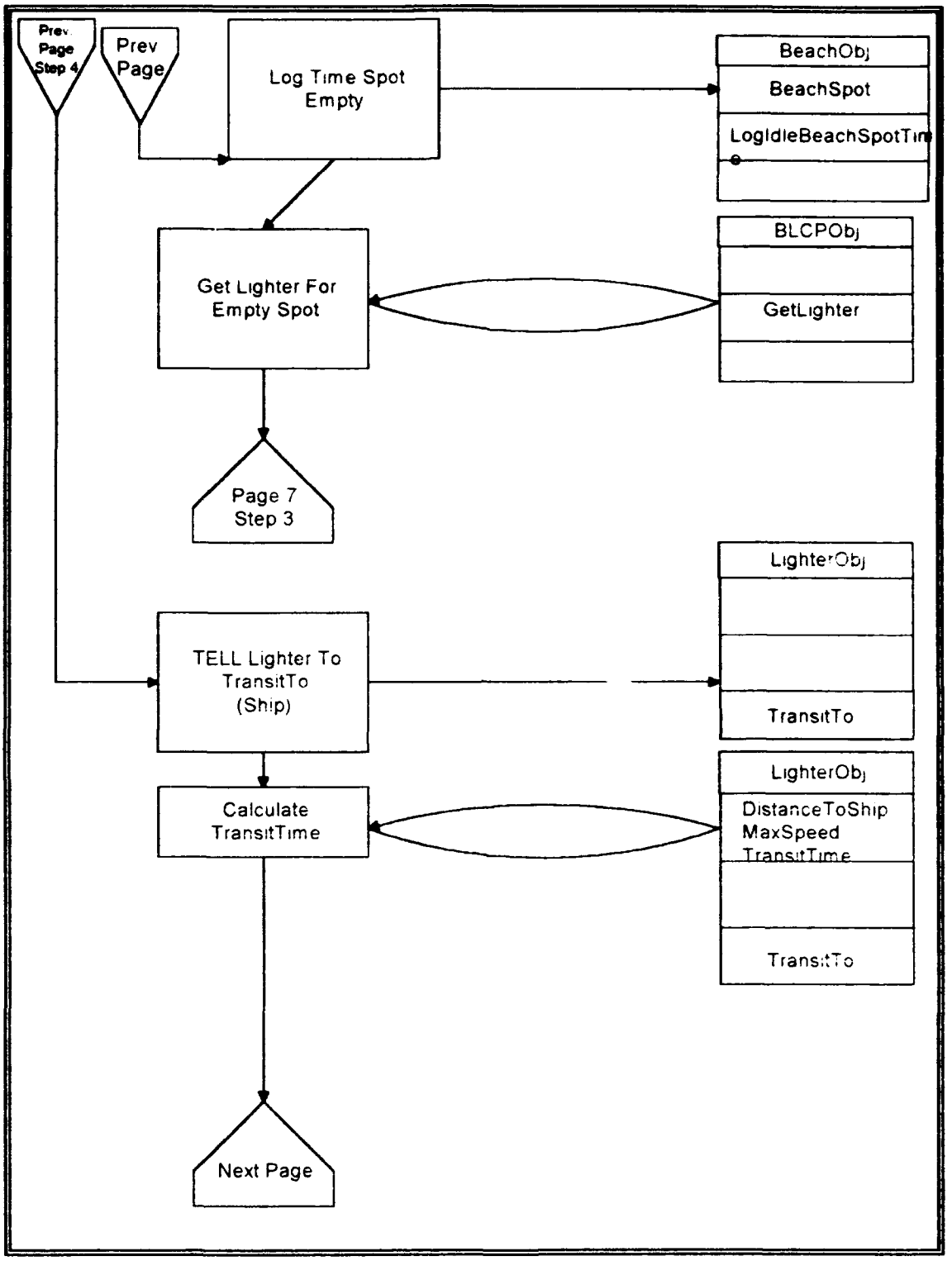

Figure 18: OOS-Pics Page 12.

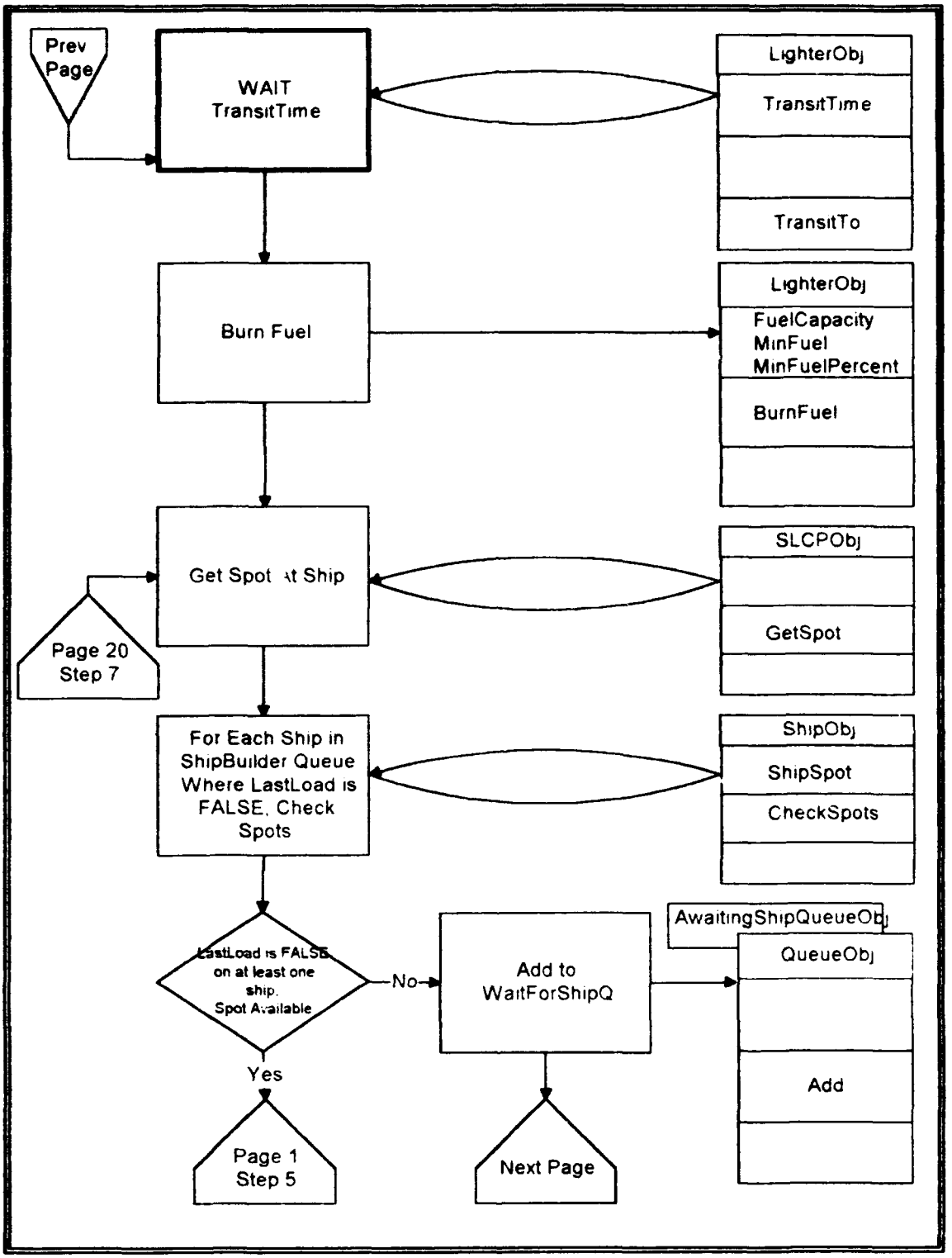

Figure 19: OOS-Pics Page 13.

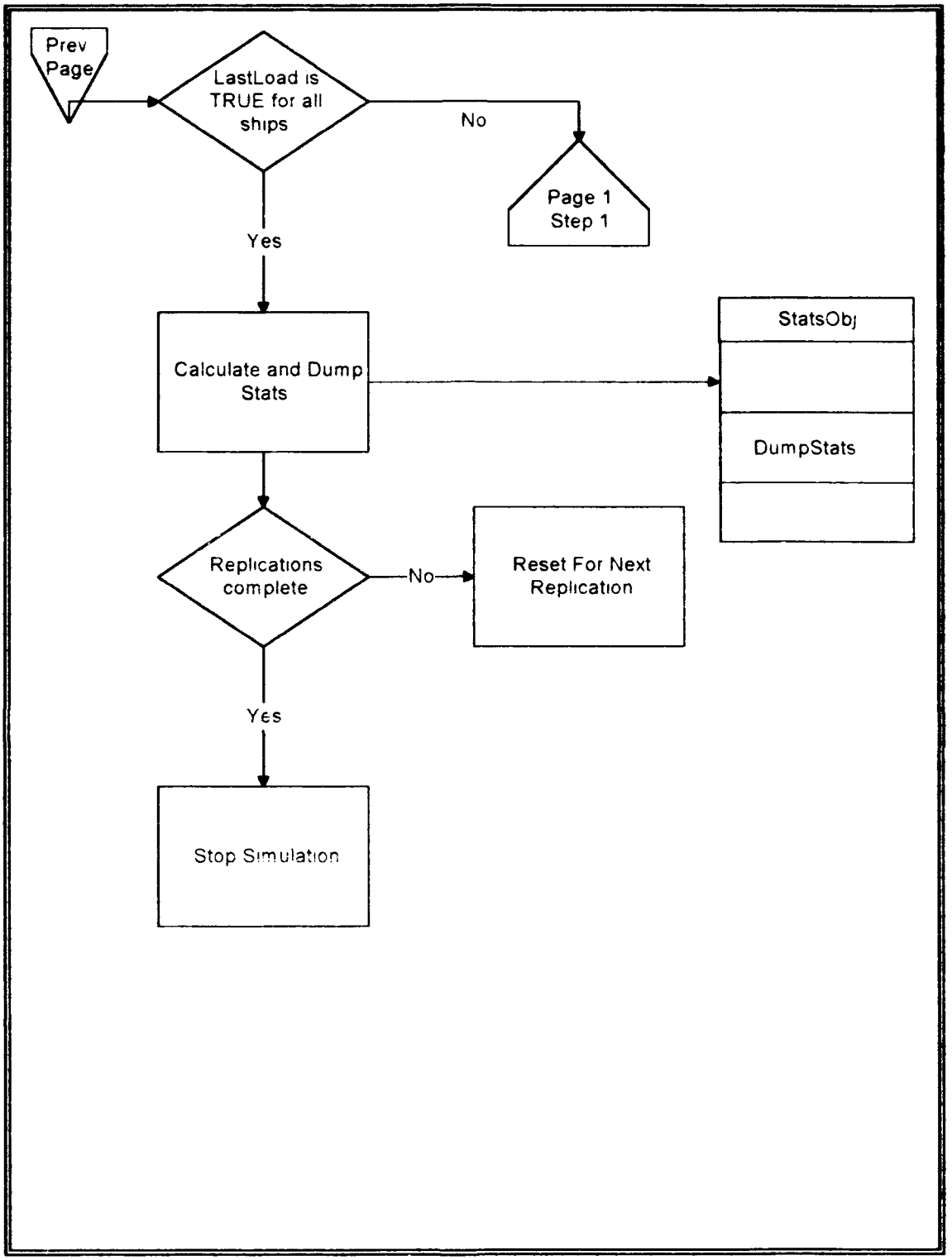

Figure 20: OOS-Pics Page 14.

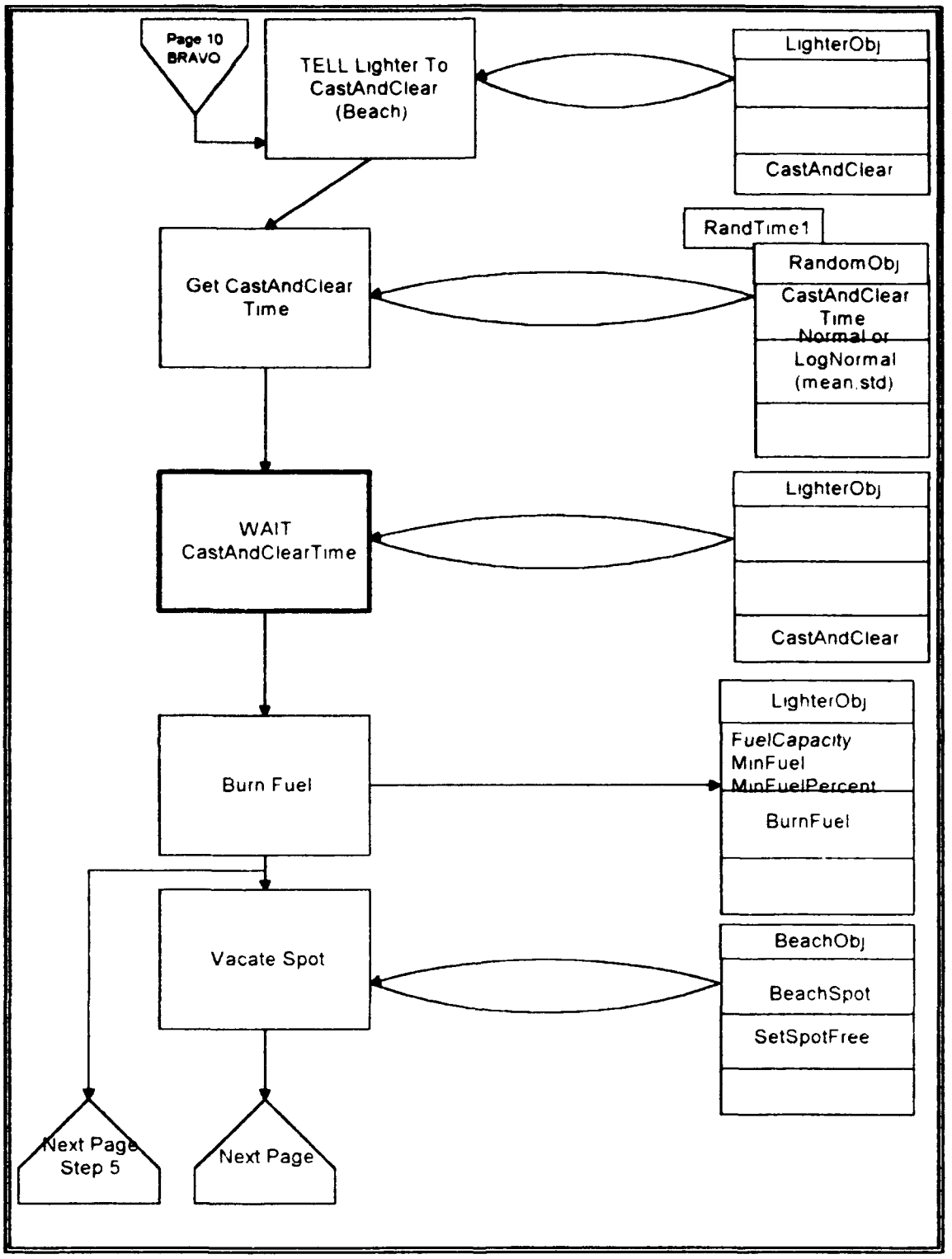

Figure 21: OOS-Pics Page 15.

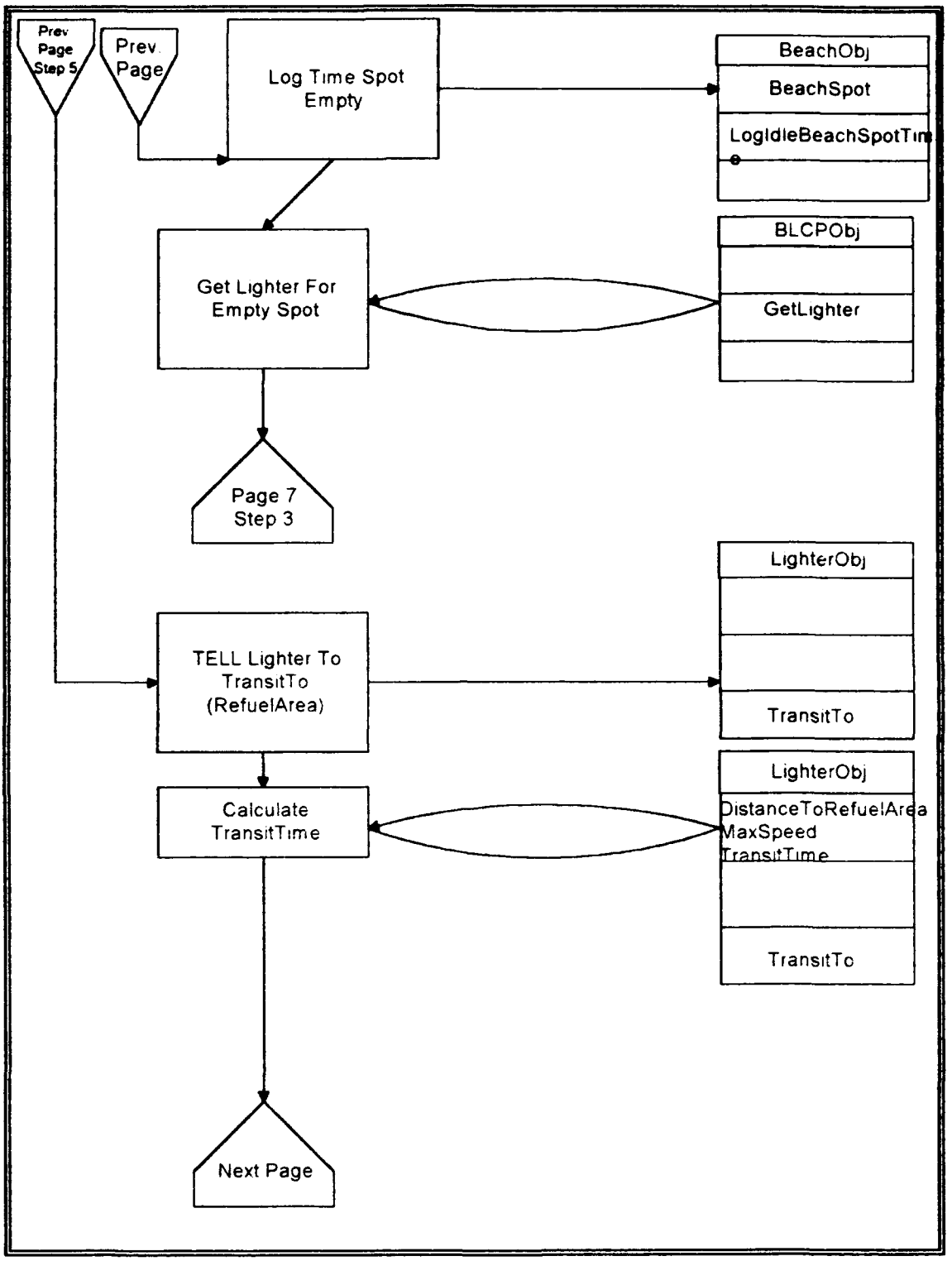

Figure 22: OOS-Pics Page 16.

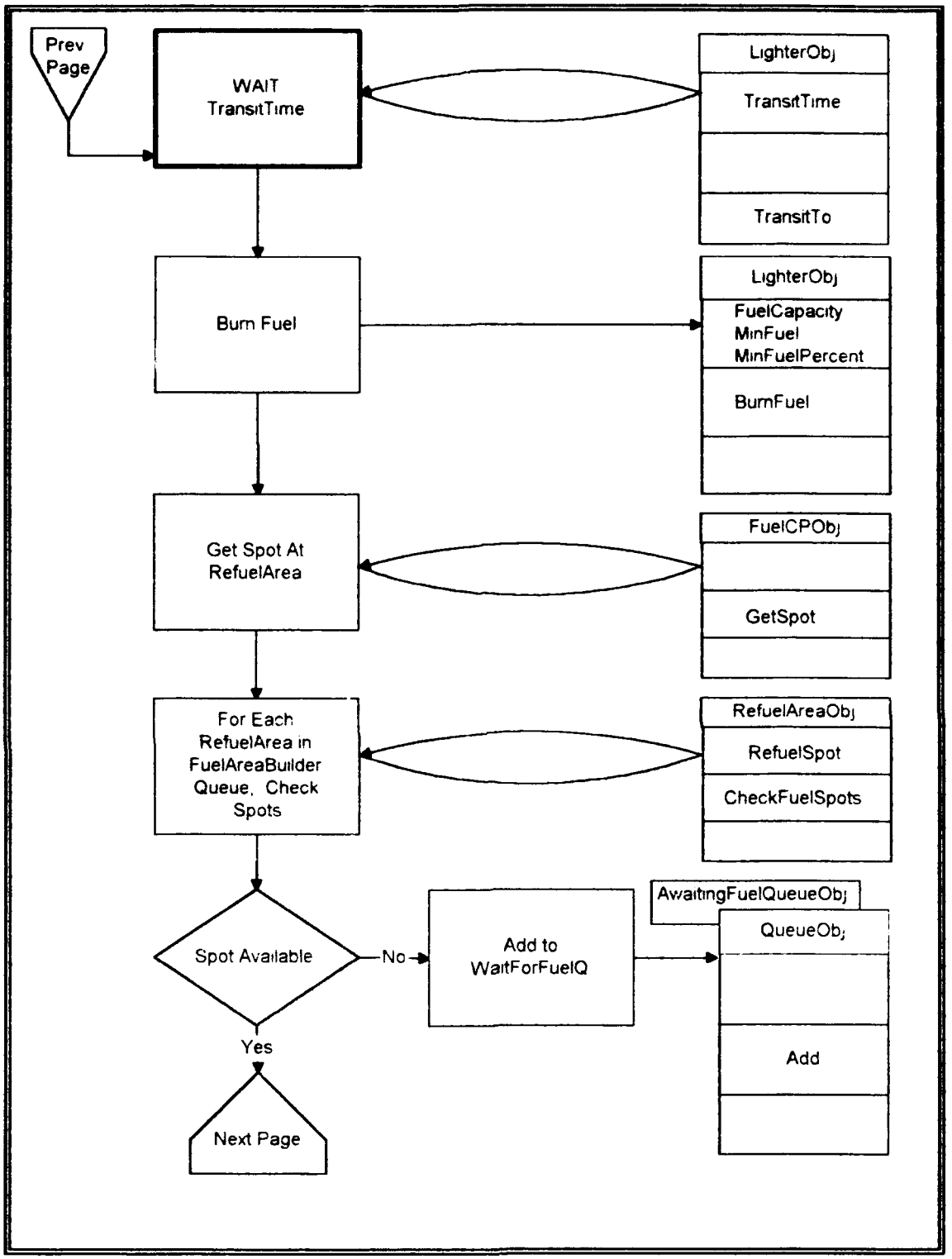

Figure 23: OOS-Pics Page 17.

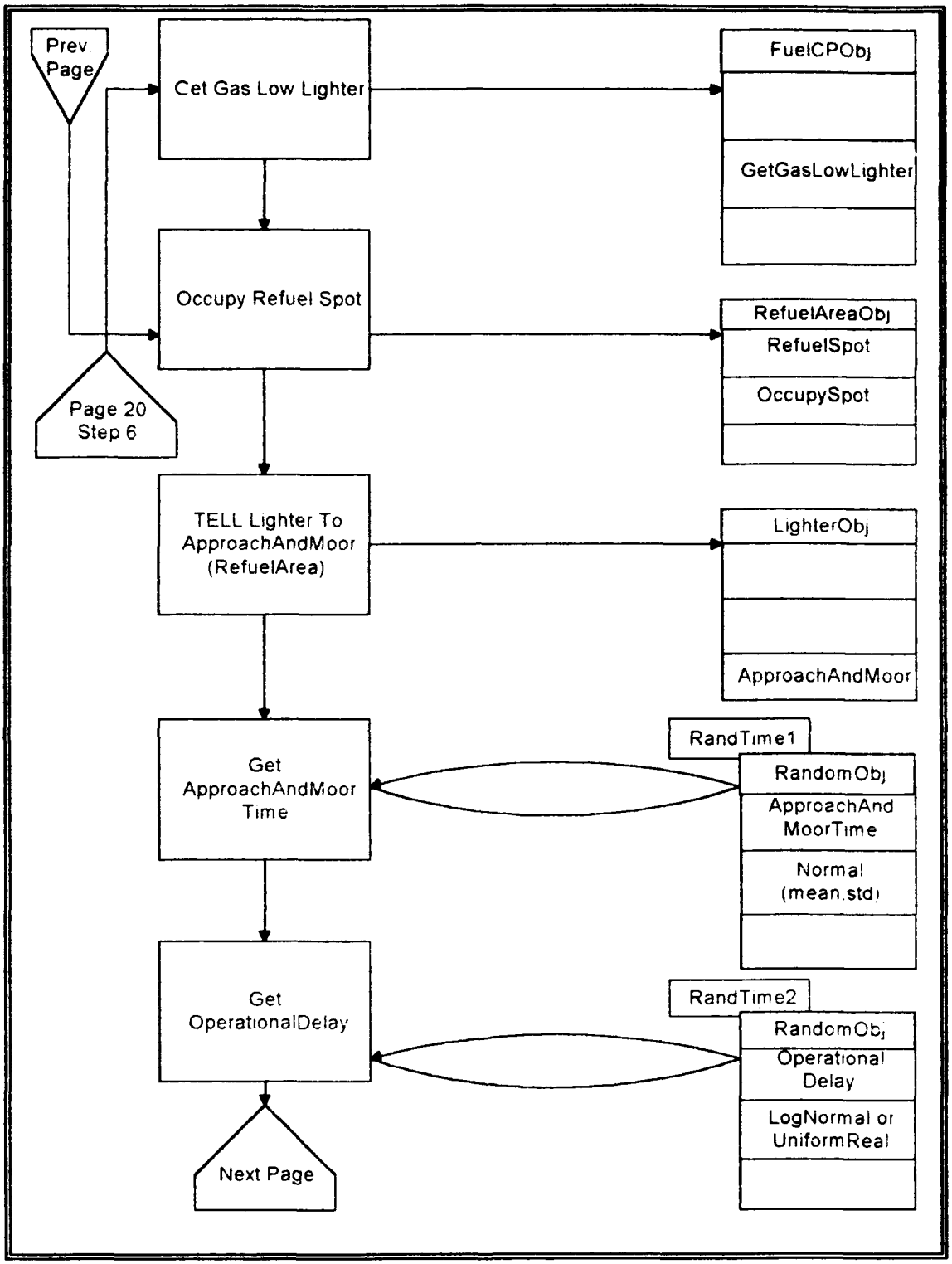

Figure 24: OOS-Pics Page 18.

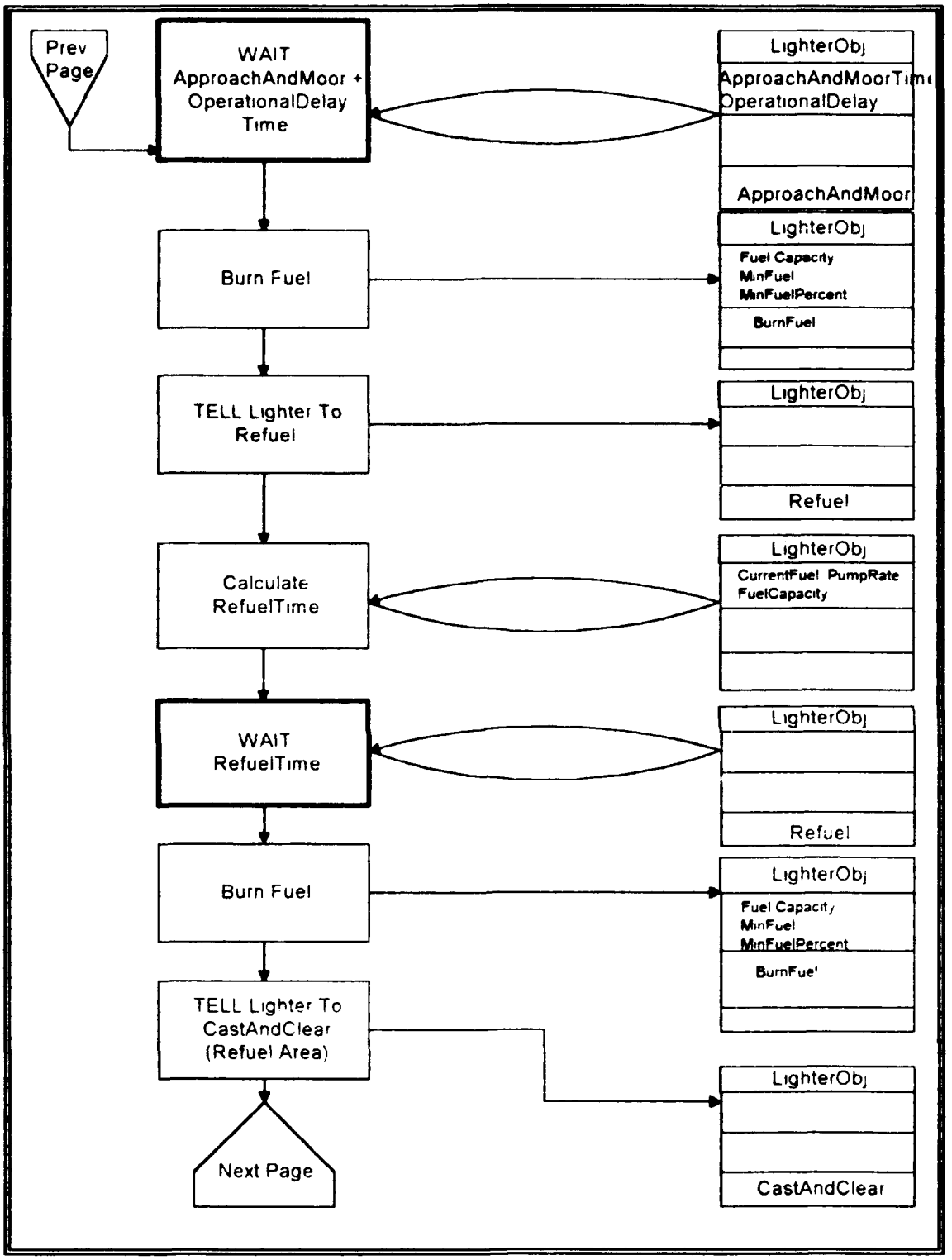

Figure 25: OOS-Pics Page 19.

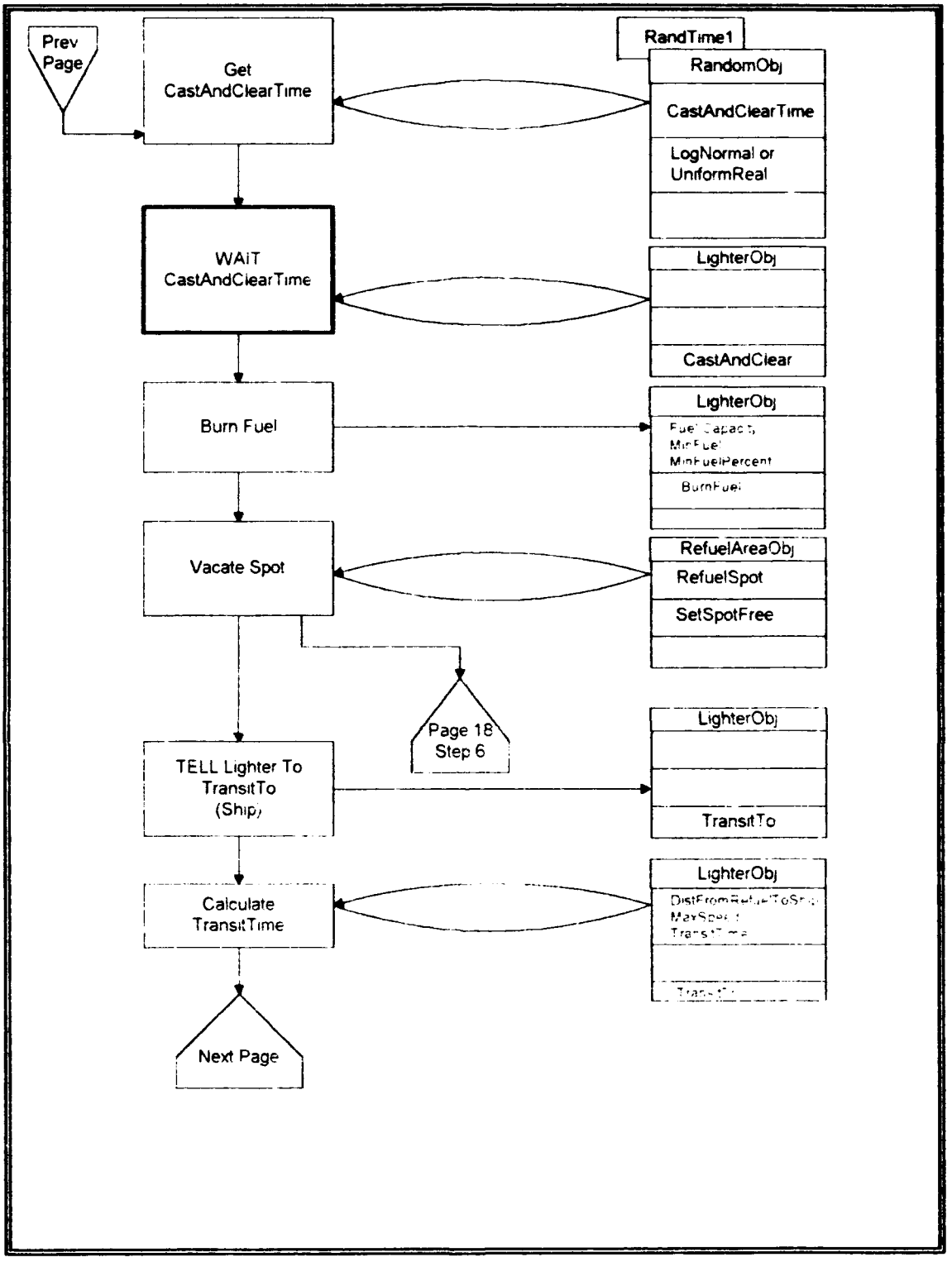

Figure 26: OOS-Pics Page 20.

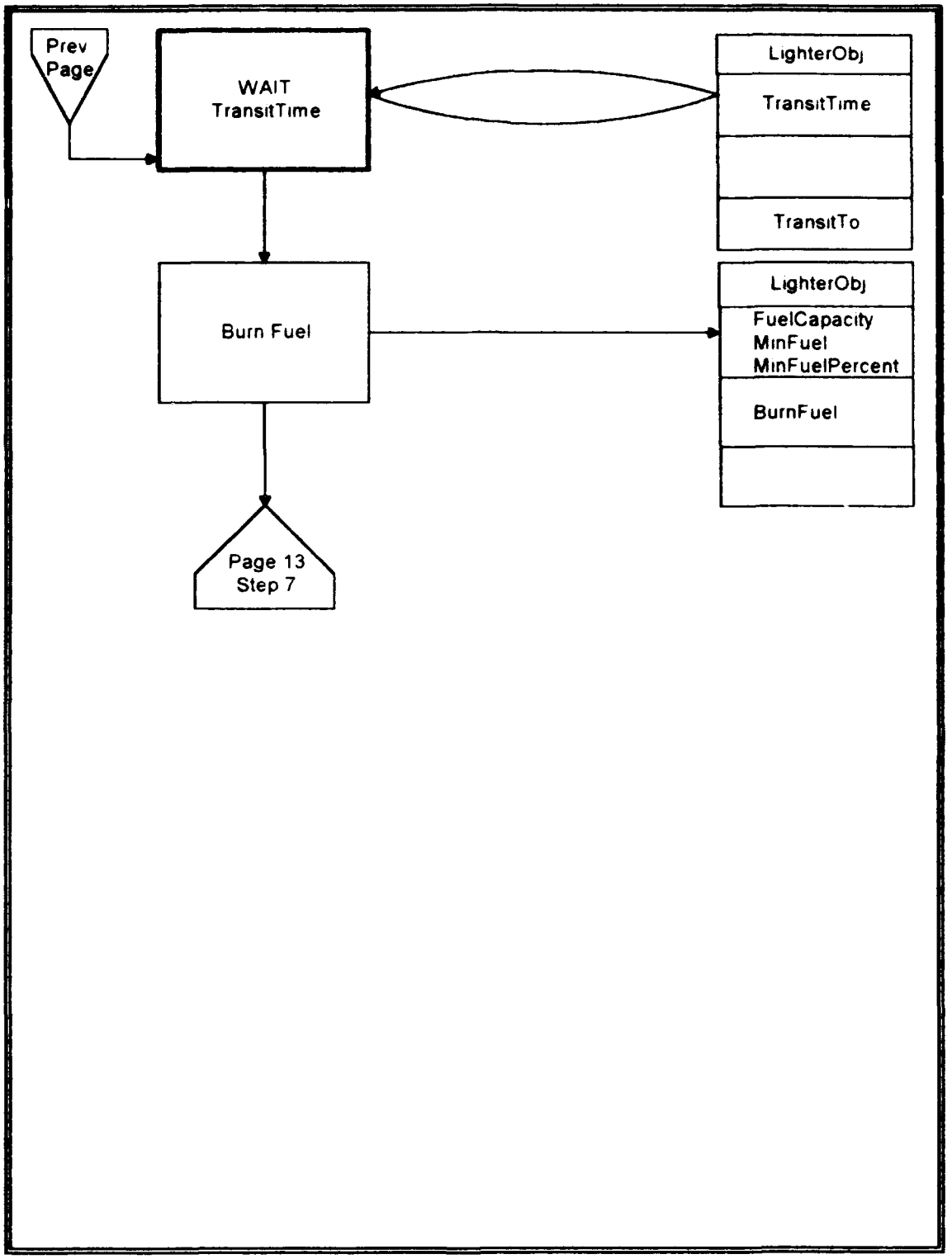

Figure 27: OOS-Pics Page 21.

APPENDIX C RO/RO OFFLOAD MODEL SOURCE CODE

```
MAIN MODULE RoRoOff;
{-------------------------------------------------------------

 Module Name: RoRoOff                    Modified: 26 Jul 93
 Author:      J. S. Noel
              Lt.  USN

DESCRIPTION:  JLOTS RoRo offload model.  Simulates the
instream offload of rolling stock from self-sustaining and
non-self-sustaining Roll On / Roll Off ships equiped for
operations using a RoRo discharge facility (RRDF) and/or
Lift On / Lift Off (LoLo) operations.  By means of input
files the user is able to change the scenario by altering
the number and type of Beach landing areas, the number and
type of lighterage to be used, the type of ship, and the
characteristics of the refueling facility.
-------------------------------------------------------------}
FROM RepMngr  IMPORT RepManager;
FROM ListAll  IMPORT ListMaster;
FROM Builder IMPORT ObjectBuilder;

BEGIN

NEW(RepManager);

NEW(ListMaster);
ASK ListMaster TO ReadAllData;

NEW(ObjectBuilder);
ASK ObjectBuilder TO BuildObjects;

ASK RepManager TO ChangeRunParms;
ASK RepManager TO Replicate;

END MODULE.
```

```
DEFINITION MODULE ListAll;

{-------------------------------------------------------------

 Module Name: ListAll                  Modified: 26 Jul 93
 Author:      M. Bailey                Modified By: J. S. Noel
              Prof. NPG                             Lt.  USN

DESCRIPTION:  Defines the ListMasterObj which NEWs the
appropriate objects and fires the methods to read in the
data files so that ship, lighter, and beach objects can be
built.
-------------------------------------------------------------}

TYPE


ListMasterObj = OBJECT

   ASK METHOD ReadAllData;

   ASK METHOD ReadShipList;
   ASK METHOD ReadLighterList;
   ASK METHOD ReadBeachList;
   ASK METHOD ReadFuelAreaList;

END OBJECT;

VAR

ListMaster : ListMasterObj;

END MODULE.
```

```
IMPLEMENTATION MODULE ListAll;

{-------------------------------------------------------------

 Module Name: ListAll                Last Modified: 26 Jul 93
 Author:      M. Bailey              Modified By: J. S. Noel
              Prof. NPG                           Lt.  USN

DESCRIPTION:  Implements the ListMasterObj which NEWs the
appropriate objects and fires the methods to read in the
data files so that ship, lighter, and beach objects can be
built.
-------------------------------------------------------------}
FROM Builder   IMPORT ShipBuilder, BeachBuilder,
                      LighterBuilder,
                      FuelAreaBuilder;
FROM ShpList   IMPORT MasterShipTypeList;
FROM LtList    IMPORT MasterLighterTypeList;
FROM BchList   IMPORT MasterBeachTypeList;
FROM RFAList   IMPORT MasterRefuelTypeList;
FROM ShpName   IMPORT MasterShipNameList, ShipNameRecType;
FROM LtName    IMPORT MasterLighterNameList,
                      LighterNameRecType;
FROM BchName   IMPORT MasterBeachNameList, BeachNameRecType;
FROM RFAName   IMPORT MasterRefuelNameList,
                      RefuelNameRecType;
FROM WriteLine IMPORT WriteLine;


OBJECT ListMasterObj;

{-------------------------------------------------------------}
ASK METHOD ReadAllData;
{-------------------------------------------------------------}

BEGIN

OUTPUT("Reading ship data ");
ASK SELF TO ReadShipList;
OUTPUT("Reading Lighter data ");
```

```
ASK SELF TO ReadLighterList;
OUTPUT("Reading Beach data ");
ASK SELF TO ReadBeachList;
OUTPUT("Reading RefuelArea data ");
ASK SELF TO ReadFuelAreaList;

END METHOD;

{-------------------------------------------------------------}
ASK METHOD ReadShipList;
{-------------------------------------------------------------}

VAR

Rec : ShipNameRecType;

BEGIN
WriteLine(" ");
WriteLine("      Read ShipNameList ");
WriteLine(" ");

NEW(MasterShipNameList);
ASK MasterShipNameList TO ReadShipNames;

NEW(MasterShipTypeList);

Rec := ASK MasterShipNameList First();
WHILE Rec <> NILREC
   OUTPUT("Reading ship info for " + Rec.ShipName);
   ASK MasterShipTypeList TO ReadShips(Rec.ShipName);

Rec := ASK MasterShipNameList Next(Rec);
END WHILE;

END METHOD;
```

```
{-------------------------------------------------------------}
ASK METHOD ReadLighterList;
{-------------------------------------------------------------}


VAR

Rec : LighterNameRecType;

BEGIN
WriteLine(" ");
WriteLine("      Read LighterNameList ");
WriteLine(" ");

NEW(MasterLighterNameList);
ASK MasterLighterNameList TO ReadLighterNames;

NEW(MasterLighterTypeList);

Rec := ASK MasterLighterNameList First();
WHILE Rec <> NILREC
   OUTPUT("Reading Lighter info for " + Rec.LighterName);
   ASK MasterLighterTypeList TO
ReadLighters(Rec.LighterName);

Rec := ASK MasterLighterNameList Next(Rec);
END WHILE;

END METHOD;


{-------------------------------------------------------------}
ASK METHOD ReadBeachList;
{-------------------------------------------------------------}


VAR

Rec : BeachNameRecType;
```

```
BEGIN
WriteLine(" ");
WriteLine("      Read BeachNameList ");
WriteLine(" ");

NEW(MasterBeachNameList);
ASK MasterBeachNameList TO ReadBeachNames;

NEW(MasterBeachTypeList);

Rec := ASK MasterBeachNameList First();
WHILE Rec <> NILREC
   OUTPUT("Reading Beach info for " + Rec.BeachName);
   ASK MasterBeachTypeList TO ReadBeaches(Rec.BeachName);

Rec := ASK MasterBeachNameList Next(Rec);
END WHILE;

END METHOD;


{-------------------------------------------------------------}
ASK METHOD ReadFuelAreaList;
{-------------------------------------------------------------}


VAR

Rec : RefuelNameRecType;

BEGIN
WriteLine(" ");
WriteLine("      Read RefuelNameList ");
WriteLine(" ");

NEW(MasterRefuelNameList);
ASK MasterRefuelNameList TO ReadRefuelNames;

NEW(MasterRefuelTypeList);
```

```
Rec := ASK MasterRefuelNameList First();
WHILE Rec <> NILREC
   OUTPUT("Reading Refuel info for " + Rec.RefuelName);
   ASK MasterRefuelTypeList TO
ReadRefuelArea(Rec.RefuelName);

Rec := ASK MasterRefuelNameList Next(Rec);
END WHILE;

END METHOD;

END OBJECT;

END MODULE.


DEFINITION MODULE RecIOHandle;

{-------------------------------------------------------------
Module Name: RecIOHandle              Last Modified: 18 Jun 93
Author:      M. Bailey                Modified By; J. S. Noel
             Prof.  NPGS                           Lt.    USN

DESCRIPTION:  Defines the RecIOHandleObj.  This PROTO object
serves as a platform for input/output objects in the
simulation.  The ProduceRec method is the key method which
inheriting objects must override.
-------------------------------------------------------------}
FROM IOMod     IMPORT StreamObj;
FROM WriteLine IMPORT WriteLine;


TYPE

SArrayType = ARRAY INTEGER OF STRING;

SHRecType = RECORD
   TopString : STRING;
   OwnedString : SArrayType;
END RECORD;
```

```
SHArrayType = ARRAY INTEGER OF SHRecType;

RecIOHandleObj = PROTO

   numberIn : INTEGER;
   ASK METHOD ReadRecs(IN FileName : STRING);
   ASK METHOD ProduceRec(IN HeadString : STRING): #ANYREC;

   {Must inherit and override to use ProduceRec.  Should
    tailor ProduceRec to meet the record type spec. of
    your application.}

   ASK METHOD ProduceRecByIndex(IN Index : INTEGER):
                                #ANYREC;

   {Produces the record by its index in SHArray.  Used
    usually  when the entire set of records is going to be
    produced at once.  No override needed, as it relies on
    ProduceRec.}

   {FindSHRec newly public}

   ASK METHOD FindSHRec(IN TopString : STRING;
                        OUT SHRec : SHRecType);

   SHArray : SHArrayType;

   ASK METHOD ReadSH(IN File : StreamObj;
                     OUT SHRec : SHRecType;
                     OUT error : BOOLEAN);

   ASK METHOD DumpRec(IN SHRec : SHRecType);

{ Dumps the rec to sim.out }
   ASK METHOD ObjTerminate;

END PROTO;

END MODULE.
```

```
IMPLEMENTATION MODULE RecIOHandle;

{-------------------------------------------------------------

Module Name: RecIOHandle          Last Modified: 18 Jun 93
Author:      M. Bailey            Modified By; J. S. Noel
             Prof.  NPGS                       Lt.    USN

DESCRIPTION:  Implements the RecIOHandleObj.  This PROTO
object serves as a platform for input/output objects in the
simulation.  The ProduceRec method is the key method which
inheriting objects must override.
-------------------------------------------------------------}
FROM IOMod     IMPORT StreamObj, FileUseType(Input);
FROM IOMod     IMPORT ReadKey;
FROM WriteLine IMPORT WriteLine;

CONST

HeadStringSpew = FALSE;
Spew = FALSE;
LittleSpew = FALSE;

TYPE

StringRecType = RECORD
   String : STRING;
   Next : StringRecType;
END RECORD;

VAR

NameOfFile : STRING;

PROTO RecIOHandleObj;

{-------------------------------------------------------------}
ASK METHOD DumpRec(IN SHRec : SHRecType);
{ Dumps the rec to sim.out }
{-------------------------------------------------------------}
```

```
VAR

i : INTEGER;
max : INTEGER;

BEGIN

WriteLine(SHRec.TopString + " ->          (has " +
          INTTOSTR(HIGH(SHRec.OwnedString))
          + " fields)");
max := HIGH(SHRec.OwnedString);
FOR i := 1 TO max
   WriteLine("    ->" + SHRec.OwnedString[i] + "<-");

END FOR;

END METHOD;


{-------------------------------------------------------------}
ASK METHOD ProduceRec(IN HeadString : STRING): ANYREC;
{-------------------------------------------------------------}

{This method does nothing, as it needs to be overridden. }

BEGIN

RETURN(NILREC);

END METHOD;


{-------------------------------------------------------------}
ASK METHOD ProduceRecByIndex(IN Index : INTEGER): ANYREC;
{-------------------------------------------------------------}

VAR

Rec : ANYREC;
```

```
BEGIN

IF Index <= HIGH(SHArray)
   Rec := ProduceRec(SHArray[Index].TopString);

ELSE
   Rec := NILREC;

END IF;

RETURN(Rec);

END METHOD;


{-------------------------------------------------------------}
ASK METHOD ReadRecs(IN FileName : STRING);
{-------------------------------------------------------------}

VAR

File : StreamObj;
numberOfSH : INTEGER;
i : INTEGER;
error : BOOLEAN;
string : STRING;

BEGIN

NameOfFile := FileName;
{NameOfFile is a local var used in error message.}

NEW(File);
ASK File TO Open(FileName, Input);

ASK File TO ReadInt(numberOfSH);
ASK File TO ReadLine(string);

numberIn := numberOfSH;
```

```
IF LittleSpew
   WriteLine("Opened file " + FileName + " which has " +
             INTTOSTR(numberOfSH) +
             " SHRecords");

END IF;

NEW(SHArray, 1..numberOfSH);
FOR i := 1 TO numberOfSH
   IF HeadStringSpew
      WriteLine("----------Rec " + INTTOSTR(i));
      OUTPUT("Rec " + INTTOSTR(i) + "  ");
   END IF;

   ReadSH(File, SHArray[i], error);
   IF error
      OUTPUT("problem reading file ", FileName, " BAD FORMAT
             DETECTED at " +
             INTTOSTR(i));
      WriteLine("problem reading file " + FileName + " BAD
                FORMAT DETECTED at " +
                INTTOSTR(i));
   END IF;

END FOR;

END METHOD;

{======================Private Methods====================}

{---------------------------------------------------------}
ASK METHOD ReadSH(IN File : StreamObj;
                  OUT SHRec : SHRecType;
                  OUT error : BOOLEAN);
{---------------------------------------------------------}

VAR

string : STRING;
junk : STRING;
```

```
numberOfStrings : INTEGER;
StringRec, OldStringRec : StringRecType;
first : StringRecType;
arrow : STRING;
stringRec : StringRecType;
i : INTEGER;
z : CHAR;

BEGIN

NEW(SHRec);
REPEAT
   ASK File TO ReadString(string);
   IF ((string = "..") OR (SUBSTR(1,1,string) = "#"))
      ASK File TO ReadLine(junk);
   END IF;

UNTIL ((string <> "..") AND (SUBSTR(1,1,string) <> "#"));

SHRec.TopString := string;

IF HeadStringSpew
   OUTPUT(SHRec.TopString);
   WriteLine("    " + SHRec.TopString);

END IF;

NEW(StringRec);
numberOfStrings := 1;
first := StringRec;

ASK File TO ReadString(arrow);
IF arrow <> "->"
   OUTPUT("file not formatted correctly");
   error := TRUE;
   RETURN;

ELSE
   error := FALSE;
```

```
END IF;

WHILE string <> "\\"
   ASK File TO ReadString(string);
   IF ((string = "..") OR (SUBSTR(1,1,string) = "#"))
      ASK File TO ReadLine(string);

   ELSE
      OldStringRec := StringRec;
      StringRec.String := string;
      NEW(StringRec);
      OldStringRec.Next := StringRec;
      numberOfStrings := numberOfStrings + 1;

   END IF;

END WHILE;

ASK File TO ReadLine(string);

IF (numberOfStrings > 0) AND NOT error
   NEW(SHRec.OwnedString, 1..numberOfStrings - 2);
   stringRec := first;

   FOR i := 1 TO numberOfStrings - 2
      SHRec.OwnedString[i] := stringRec.String;

      IF Spew
         OUTPUT(i, "  ", stringRec.String);
         WriteLine(SHRec.OwnedString[i] + "   ");

      END IF;

      stringRec := stringRec.Next;

   END FOR;

END IF;
```

```
END METHOD;


{-------------------------------------------------------------}
ASK METHOD FindSHRec(IN TopString : STRING;
                     OUT SHRec : SHRecType);
{-------------------------------------------------------------}

VAR

ThisRec : SHRecType;
i : INTEGER;

BEGIN

i := 0;

REPEAT
   INC(i);
   ThisRec := SHArray[i];

UNTIL ((i >= HIGH(SHArray)) OR (ThisRec.TopString =
        TopString));

IF (ThisRec. TopString = TopString)
   SHRec := ThisRec;

ELSE
   SHRec := NILREC;
   WriteLine("FindSHRec of RecIOHandle came up empty
             searching for TopString " +
             TopString);
   WriteLine("while looking in file " + NameOfFile);

END IF;

IF Spew
   WriteLine(" !!!  find sh rec called with topstring " +
             TopString +
```

```
            " and returns rec with topstring " +
            ThisRec.TopString);

END IF;


END METHOD;

{-------------------------------------------------------------}
ASK METHOD ObjTerminate;
{-------------------------------------------------------------}

VAR

i : INTEGER;
REC : SHRecType;

BEGIN

FOR i := 1 TO HIGH(SHArray)
   REC := SHArray[i];
   DISPOSE(REC.OwnedString);
   DISPOSE(REC);

END FOR;

END METHOD;

END PROTO;

END MODULE.



DEFINITION MODULE ShpName;

{-------------------------------------------------------------

 Module Name: ShpName                Last Modified: 28 Jul 93
 Author:      J. S. Noel
```

```
                Lt.     USN

DESCRIPTION:  Implements the ShipNameListObj and the
ShipNameIORecHandleObj which together provide the means for
inputing the Ship Names into the simulation for a given
scenario.
-------------------------------------------------------------}


FROM RecIOHandle IMPORT RecIOHandleObj;
FROM ListMod     IMPORT QueueList;

TYPE

ShipNameRecType = RECORD
   ShipName          : STRING;

END RECORD;

ShipNameListObj = OBJECT(QueueList[ANYREC :ShipNameRecType])
   ASK METHOD ReadShipNames;
END OBJECT;

ShipNameIORecHandleObj=OBJECT(RecIOHandleObj[ANYREC:
                                ShipNameRecType])
END OBJECT;

VAR

ShipNameIOHandler : ShipNameIORecHandleObj;
MasterShipNameList : ShipNameListObj;

END MODULE.
```

```
IMPLEMENTATION MODULE ShpName;

{-------------------------------------------------------------

 Module Name: ShpName                  Last Modified: 28 Jul 93
 Author:      J. S. Noel
              Lt.    USN

DESCRIPTION:  Implements the ShipNameListObj and the
ShipNameIORecHandleObj which together provide the means for
inputing the Beach names into the simulation for a given
scenario.
-------------------------------------------------------------}
FROM RecIOHandle IMPORT SHArrayType;


OBJECT ShipNameListObj;

{-------------------------------------------------------------}
ASK METHOD ReadShipNames;
{-------------------------------------------------------------}

VAR
Rec      : ShipNameRecType;
index    : INTEGER;
high     : INTEGER;
SHArray  : SHArrayType;


BEGIN

IF ShipNameIOHandler = NILOBJ
   NEW(ShipNameIOHandler);
   ASK ShipNameIOHandler TO ReadRecs("ShpName.dat");
END IF;

SHArray := ASK ShipNameIOHandler SHArray;
high := HIGH(SHArray);

FOR index := 1 TO high
```

```
   NEW(Rec);
   Rec.ShipName := SHArray[index].TopString;
   Add(Rec);

END FOR;

END METHOD;

END OBJECT;

END MODULE.



DEFINITION MODULE ShpList;

{-------------------------------------------------------------
Module Name: ShpList                    Last Modified: 18 Jun 93
Author:  J. S. Noel
         Lt.    USN

DESCRIPTION:  Defines the ShipTypeListObj and the
ShipTypeIORecHandleObj which together provide the means for
inputing the RoRo's into the simulation for a given
scenario.
-------------------------------------------------------------}
FROM RecIOHandle IMPORT RecIOHandleObj;
FROM ListMod     IMPORT QueueList;
FROM Global      IMPORT SpotRecType;
FROM Ship        IMPORT ShipTypeType;

TYPE

SpotArrayType = ARRAY INTEGER OF SpotRecType;

ShipTypeRecType = RECORD

   ShipTypeName          : STRING;
   TypeShip              : ShipTypeType;
   DistanceFromBLCP      : REAL;
```

```
   NumOfSpots              : INTEGER;
   SpotArray               : SpotArrayType;
   NumOfLoLoVehicles       : INTEGER;
   NumOfRRDFVehicles       : INTEGER;
   NumOfAnySpotVehicles    : INTEGER;


END RECORD;

ShipTypeListObj = OBJECT(QueueList[ANYREC :
                          ShipTypeRecType])
   ASK METHOD ReadShips(IN ShipName : STRING);
END OBJECT;

ShipTypeIORecHandleObj = OBJECT(RecIOHandleObj[ANYREC :
                                 ShipTypeRecType])
   OVERRIDE
   ASK METHOD ProduceRec(IN HeadString : STRING):
                         ShipTypeRecType;
END OBJECT;

VAR

ShipTypeIOHandler : ShipTypeIORecHandleObj;
MasterShipTypeList : ShipTypeListObj;

END MODULE.



IMPLEMENTATION MODULE ShpList;

{-------------------------------------------------------------
Module Name: ShpList                    Last Modified: 18 Jun 93
Author:  J. S. Noel
         Lt.    USN

DESCRIPTION:  Implements the ShipTypeListObj and the
ShipTypeIORecHandleObj which together provide the means for
```

```
inputing the RoRo's into the simulation for a given
scenario.
-------------------------------------------------------------}
FROM RecIOHandle IMPORT SHRecType;
FROM WriteLine   IMPORT WriteLine;
FROM Global      IMPORT SpotRecType;
FROM Convert     IMPORT SpotTypeToStr,StrToSpotType,
                        ShipTypeToStr,StrToShipType,
                        BooleanToStr;

OBJECT ShipTypeIORecHandleObj;

{-------------------------------------------------------------}
ASK METHOD ProduceRec(IN HeadString : STRING):
                      ShipTypeRecType;
{-------------------------------------------------------------}

VAR
SHRec : SHRecType;
Rec : ShipTypeRecType;
index, i : INTEGER;
Char : CHAR;
Spot : SpotRecType;

BEGIN
WriteLine("  producing record Ship type " + HeadString);
OUTPUT("  producing record Ship type " + HeadString);
FindSHRec(HeadString, SHRec);
WriteLine(" ");
IF SHRec = NILREC
   OUTPUT("No record Found of " + HeadString);
   WriteLine("No record Found of " + HeadString);
   HALT;
END IF;
OUTPUT(" got SHRec");
NEW(Rec);

Rec.ShipTypeName := SHRec.TopString;
OUTPUT("ShipTypeName is " + Rec.ShipTypeName);
WriteLine("ShipTypeName is " + Rec.ShipTypeName);
```

```
index := 1;

Rec.TypeShip := StrToShipType(SHRec.OwnedString[index]);
WriteLine("TypeShip is " + ShipTypeToStr(Rec.TypeShip));
INC(index);


Rec.DistanceFromBLCP := STRTOREAL(SHRec.OwnedString[index]);
WriteLine("DistanceFromBLCP is " +
REALTOSTR(Rec.DistanceFromBLCP));
INC(index);

Rec.NumOfSpots := STRTOINT(SHRec.OwnedString[index]);
WriteLine("NumOfSpots is " + INTTOSTR(Rec.NumOfSpots));
INC(index);

NEW(Rec.SpotArray, 1..Rec.NumOfSpots);
OUTPUT("index = " + INTTOSTR(index));

IF (Rec.NumOfSpots > 0)
   i := 1;
   WHILE i <= Rec.NumOfSpots
      NEW(Spot);
      Spot.SpotClassification :=
StrToSpotType(SHRec.OwnedString[index]);

      WriteLine("Spot "+ INTTOSTR(i) + " is Type " +
                SpotTypeToStr(Spot.SpotClassification));

      OUTPUT("Spot "+ INTTOSTR(i) + " is Type " +
             SpotTypeToStr(Spot.SpotClassification));

      Char := SCHAR(SHRec.OwnedString[index+1], 1);
      CASE Char
        WHEN 'T','t' : Spot.SpotFree := TRUE;
        WHEN 'F','f' : Spot.SpotFree := FALSE;
      END CASE;
      WriteLine("Spot "+ INTTOSTR(i) + " is Free (T/F) "  +
                BooleanToStr(Spot.SpotFree));
```

```
      OUTPUT("Spot "+ INTTOSTR(i) + " is Free (T/F) " +
Char);

      Rec.SpotArray[i] := Spot;
      i := i + 1;
      index := index + 2;
   END WHILE;
END IF;

index := index;

Rec.NumOfLoLoVehicles := STRTOINT(SHRec.OwnedString[index]);
WriteLine("NumOfLoLoVehicles is " +
           INTTOSTR(Rec.NumOfLoLoVehicles));
INC(index);

Rec.NumOfRRDFVehicles := STRTOINT(SHRec.OwnedString[index]);
WriteLine("NumOfRRDFVehicles is " +
           INTTOSTR(Rec.NumOfRRDFVehicles));
INC(index);

Rec.NumOfAnySpotVehicles :=
                          STRTOINT(SHRec.OwnedString[index]);
WriteLine("NumOfAnySpotVehicles is " +
           INTTOSTR(Rec.NumOfAnySpotVehicles));
INC(index);

OUTPUT("finished reading Ship type information");

RETURN(Rec);

END METHOD;

END OBJECT;


OBJECT ShipTypeListObj;
```

```
{-------------------------------------------------------------}
ASK METHOD ReadShips(IN ShipType : STRING);
{-------------------------------------------------------------}
VAR
Rec : ShipTypeRecType;

BEGIN
IF ShipTypeIOHandler = NILOBJ
   NEW(ShipTypeIOHandler);
   ASK ShipTypeIOHandler TO ReadRecs("ShipType.dat");
END IF;

OUTPUT("ship handler instanciated and full of raw records");
OUTPUT("about to produce type record for " +ShipType);
Rec := ASK ShipTypeIOHandler TO ProduceRec(ShipType);
OUTPUT(" got the record complete ");

IF (Rec <> NILREC)
   Add(Rec);
ELSE
   OUTPUT(" never found record!");
END IF;

END METHOD;

END OBJECT;

END MODULE.
```

```
DEFINITION MODULE LtName;

{-------------------------------------------------------------

 Module Name: LtName                    Last Modified: 28 Jul 93
 Author:      J. S. Noel
              Lt.    USN

DESCRIPTION:  Defines the LighterNameListObj and the
LighterNameIORecHandleObj which together provide the means
for inputing the Lighter Names into the simulation for a
given scenario.
-------------------------------------------------------------}

FROM RecIOHandle IMPORT RecIOHandleObj;
FROM ListMod     IMPORT QueueList;


TYPE

LighterNameRecType = RECORD
   LighterName          : STRING;

END RECORD;

LighterNameListObj = OBJECT(QueueList[ANYREC :
                              LighterNameRecType])
   ASK METHOD ReadLighterNames;
END OBJECT;

LighterNameIORecHandleObj=OBJECT(RecIOHandleObj[ANYREC:
                                    LighterNameRecType])
END OBJECT;

VAR

LighterNameIOHandler : LighterNameIORecHandleObj;
MasterLighterNameList : LighterNameListObj;

END MODULE.
```

```
IMPLEMENTATION MODULE LtName;

{-------------------------------------------------------------

 Module Name: LtName                    Last Modified: 28 Jul 93
 Author:      J. S. Noel
              Lt.    USN

DESCRIPTION:  Implements the LighterNameListObj and the
LighterNameIORecHandleObj which together provide the means
for inputing the Lighter Names into the simulation for a
given scenario.
-------------------------------------------------------------}

FROM RecIOHandle IMPORT SHArrayType;
FROM WriteLine   IMPORT WriteLine;


OBJECT LighterNameListObj;

{-------------------------------------------------------------}
ASK METHOD ReadLighterNames;
{-------------------------------------------------------------}

VAR
Rec      : LighterNameRecType;
index    : INTEGER;
high     : INTEGER;
SHArray  : SHArrayType;


BEGIN

IF LighterNameIOHandler = NILOBJ
   NEW(LighterNameIOHandler);
   ASK LighterNameIOHandler TO ReadRecs("LtName.dat");
END IF;

SHArray := ASK LighterNameIOHandler SHArray;
high := HIGH(SHArray);
```

```
FOR index := 1 TO high
   WriteLine("Reading Lighter Name in LtName.mod ");
   NEW(Rec);
   Rec.LighterName := SHArray[index].TopString;
   Add(Rec);

END FOR;

WriteLine("----exit ReadLighterNames-----")

END METHOD;

END OBJECT;

END MODULE.


DEFINITION MODULE LtList;

{-------------------------------------------------------------

Module Name: LtList                    Last Modified: 28 Jul 93
Author:  J. S. Noel
         Lt.    USN

DESCRIPTION:  Defines the LighterTypeListObj and the
LighterTypeIORecHandleObj which together provide the means
for inputing the Lighter's into the simulation for a given
scenario.
-------------------------------------------------------------}
FROM RecIOHandle IMPORT RecIOHandleObj;
FROM ListMod     IMPORT QueueList;
FROM Global      IMPORT SpotRecType, SpotType;
FROM Lighter     IMPORT LighterNameType;

TYPE


LighterTypeRecType = RECORD
```

```
   ID                      : STRING;
   LighterTypeName         : LighterNameType;
   SpotRequired            : SpotType;
   SpeedMax                : REAL;
   SpeedFull               : REAL;
   MaxLoad                 : INTEGER;
   FuelCap                 : REAL;
   BurnRate                : REAL;
   MinFuel                 : REAL;

END RECORD;

LighterTypeListObj = OBJECT(QueueList[ANYREC :
                              LighterTypeRecType])
   ASK METHOD ReadLighters(IN LighterName : STRING);
END OBJECT;

LighterTypeIORecHandleObj = OBJECT(RecIOHandleObj[ANYREC :
                                    LighterTypeRecType])
   OVERRIDE
   ASK METHOD ProduceRec(IN HeadString : STRING):
                          LighterTypeRecType;
END OBJECT;

VAR

LighterTypeIOHandler : LighterTypeIORecHandleObj;
MasterLighterTypeList : LighterTypeListObj;

END MODULE.
```

```
IMPLEMENTATION MODULE LtList;

{-------------------------------------------------------------
Module Name: LtList                    Last Modified: 18 Jun 93
Author:  J. S. Noel
         Lt.    USN

DESCRIPTION:  Implements the LighterTypeListObj and the
LighterTypeIORecHandleObj which together provide the means
for inputing the Lighter's into the simulation for a given
scenario.
-------------------------------------------------------------}
FROM RecIOHandle IMPORT SHRecType;
FROM WriteLine   IMPORT WriteLine;
FROM Global      IMPORT SpotRecType;
FROM Convert     IMPORT SpotTypeToStr, StrToSpotType,
                        LighterNameTypeToStr,
                        StrToLighterNameType;

OBJECT LighterTypeIORecHandleObj;

{-------------------------------------------------------------}
ASK METHOD ProduceRec(IN HeadString : STRING):
                        LighterTypeRecType;
{-------------------------------------------------------------}

VAR
SHRec : SHRecType;
Rec : LighterTypeRecType;
index, i : INTEGER;
Char : CHAR;


BEGIN
WriteLine("  producing record Lighter type " + HeadString);
OUTPUT("  producing record Lighter type " + HeadString);
FindSHRec(HeadString, SHRec);
WriteLine(" ");
IF SHRec = NILREC
   OUTPUT("No record Found of " + HeadString);
```

```
   WriteLine("No record Found of " + HeadString);
   HALT;
END IF;
OUTPUT(" got SHRec");

NEW(Rec);
Rec.ID := SHRec.TopString;
WriteLine("LighterID is " + Rec.ID);
index := 1;

Rec.LighterTypeName :=
StrToLighterNameType(SHRec.OwnedString[index]);
OUTPUT("LighterTypeName is " +
LighterNameTypeToStr(Rec.LighterTypeName));
WriteLine("LighterTypeName is " +
LighterNameTypeToStr(Rec.LighterTypeName));
INC(index);

Rec.SpotRequired := StrToSpotType(SHRec.OwnedString[index]);
WriteLine("SpotRequired is " +
SpotTypeToStr(Rec.SpotRequired));
INC(index);

Rec.SpeedMax := STRTOREAL(SHRec.OwnedString[index]);
WriteLine("MaxSpeed is " + REALTOSTR(Rec.SpeedMax));
INC(index);

Rec.SpeedFull := STRTOREAL(SHRec.OwnedString[index]);
WriteLine("Full Load Speed is " + REALTOSTR(Rec.SpeedFull));
INC(index);

Rec.MaxLoad := STRTOINT(SHRec.OwnedString[index]);
WriteLine("Max Load is " + INTTOSTR(Rec.MaxLoad));
INC(index);

Rec.FuelCap := STRTOREAL(SHRec.OwnedString[index]);
WriteLine("Fuel capacity is " + REALTOSTR(Rec.FuelCap));
INC(index);

Rec.BurnRate := STRTOREAL(SHRec.OwnedString[index]);
```

```
WriteLine("Fuel burn rate is " + REALTOSTR(Rec.BurnRate));
INC(index);

Rec.MinFuel := STRTOREAL(SHRec.OwnedString[index]);
WriteLine("minimum fuel percentage is " +
          REALTOSTR(Rec.MinFuel));
INC(index);

OUTPUT("finished reading Lighter type information");

RETURN(Rec);

END METHOD;

END OBJECT;


OBJECT LighterTypeListObj;

{-------------------------------------------------------------}
ASK METHOD ReadLighters(IN LighterType : STRING);
{-------------------------------------------------------------}

VAR
Rec : LighterTypeRecType;

BEGIN
IF LighterTypeIOHandler = NILOBJ
   NEW(LighterTypeIOHandler);
   ASK LighterTypeIOHandler TO ReadRecs("LtType.dat");
END IF;

OUTPUT("Lighter handler instanciated and full of raw
records");
OUTPUT("about to produce type record for " + LighterType);
Rec := ASK LighterTypeIOHandler TO ProduceRec(LighterType);
OUTPUT(" got the record complete ");
IF Rec <> NILREC
   Add(Rec);
ELSE
```

```
   OUTPUT(" never found record!");
END IF;

END METHOD;

END OBJECT;

END MODULE.


DEFINITION MODULE BchName;

{-------------------------------------------------------------

 Module Name: BchName                    Last Modified: 28 Jul 93
 Author:      J. S. Noel
              Lt.    USN

DESCRIPTION:  Defines the BeachNameListObj and the
BeachNameIORecHandleObj which together provide the means for
inputing the Beach Names into the simulation for a given
scenario.
-------------------------------------------------------------}

FROM RecIOHandle IMPORT RecIOHandleObj;
FROM ListMod     IMPORT QueueList;


TYPE

BeachNameRecType = RECORD
   BeachName          : STRING;

END RECORD;

BeachNameListObj =OBJECT(QueueList[ANYREC:BeachNameRecType])
   ASK METHOD ReadBeachNames;
END OBJECT;
```

```
BeachNameIORecHandleObj = OBJECT(RecIOHandleObj[ANYREC :
BeachNameRecType])
END OBJECT;

VAR

BeachNameIOHandler : BeachNameIORecHandleObj;
MasterBeachNameList : BeachNameListObj;

END MODULE.


IMPLEMENTATION MODULE BchName;

{-------------------------------------------------------------
 Module Name: BchName                Last Modified: 28 Jul 93
 Author:      J. S. Noel
              Lt.    USN

DESCRIPTION:  Implements the BeachNameListObj and the
BeachNameIORecHandleObj which together provide the means for
inputing the Beach names into the simulation for a given
scenario.
-------------------------------------------------------------}
FROM RecIOHandle IMPORT SHArrayType;


OBJECT BeachNameListObj;

{-------------------------------------------------------------}
ASK METHOD ReadBeachNames;
{-------------------------------------------------------------}

VAR
Rec      : BeachNameRecType;
index    : INTEGER;
high     : INTEGER;
SHArray  : SHArrayType;
```

```
BEGIN

IF BeachNameIOHandler = NILOBJ
   NEW(BeachNameIOHandler);
   ASK BeachNameIOHandler TO ReadRecs("BchName.dat");
END IF;

SHArray := ASK BeachNameIOHandler SHArray;
high := HIGH(SHArray);

FOR index := 1 TO high
   NEW(Rec);
   Rec.BeachName := SHArray[index].TopString;
   Add(Rec);

END FOR;

END METHOD;

END OBJECT;

END MODULE.


DEFINITION MODULE BchList;

{-------------------------------------------------------------

Module Name: BchList                    Last Modified: 28 Jul 93
Author:  J. S. Noel
         Lt.    USN

DESCRIPTION:  Defines the BeachTypeListObj and the
BeachTypeIORecHandleObj which together provide the means for
inputing the Beaches into the simulation for a given
scenario.
-------------------------------------------------------------}
FROM RecIOHandle IMPORT RecIOHandleObj;
FROM ListMod     IMPORT QueueList;
FROM Global      IMPORT SpotRecType, SpotType;
```

```
FROM Beach        IMPORT BeachType;
FROM ShpList      IMPORT SpotArrayType;

TYPE


BeachTypeRecType = RECORD

   ID                      : STRING;
   BeachTypeName           : BeachType;
   NumOfSpots              : INTEGER;
   SpotArray               : SpotArrayType;
   DistanceFromSLCP        : REAL;
   DistBeachToArea         : REAL;

END RECORD;

BeachTypeListObj = OBJECT(QueueList[ANYREC :
                            BeachTypeRecType])
   ASK METHOD ReadBeaches(IN BeachName : STRING);
END OBJECT;

BeachTypeIORecHandleObj = OBJECT(RecIOHandleObj[ANYREC :
                                    BeachTypeRecType])
   OVERRIDE
   ASK METHOD ProduceRec(IN HeadString : STRING):
                           BeachTypeRecType;
END OBJECT;

VAR

BeachTypeIOHandler : BeachTypeIORecHandleObj;
MasterBeachTypeList : BeachTypeListObj;

END MODULE.
```

```
IMPLEMENTATION MODULE BchList;

{-------------------------------------------------------------
Module Name: BchList                    Last Modified: 28 Jul 93
Author:  J. S. Noel
         Lt.    USN

DESCRIPTION:  Implements the BeachTypeListObj and the
BeachTypeIORecHandleObj which together provide the means for
inputing the Beaches into the simulation for a given
scenario.
-------------------------------------------------------------}
FROM RecIOHandle IMPORT SHRecType;
FROM WriteLine   IMPORT WriteLine;
FROM Global      IMPORT SpotRecType;
FROM Convert     IMPORT SpotTypeToStr, StrToSpotType,
                        BeachTypeToStr, StrToBeachType,
                        BooleanToStr;


OBJECT BeachTypeIORecHandleObj;

{-------------------------------------------------------------}
ASK METHOD ProduceRec(IN HeadString : STRING):
                        BeachTypeRecType;
{-------------------------------------------------------------}

VAR
SHRec : SHRecType;
Rec : BeachTypeRecType;
index, i : INTEGER;
Char : CHAR;
Spot : SpotRecType;


BEGIN
WriteLine("  producing record Beach type " + HeadString);
OUTPUT("  producing record Beach type " + HeadString);
FindSHRec(HeadString, SHRec);
WriteLine(" ");
```

```
IF SHRec = NILREC
   OUTPUT("No record Found of " + HeadString);
   WriteLine("No record Found of " + HeadString);
   HALT;
END IF;
OUTPUT(" got SHRec");

NEW(Rec);
Rec.ID := SHRec.TopString;
WriteLine("BeachID is " + Rec.ID);
index := 1;

Rec.BeachTypeName :=
StrToBeachType(SHRec.OwnedString[index]);
OUTPUT("BeachTypeName is " +
BeachTypeToStr(Rec.BeachTypeName));
WriteLine("BeachTypeName is " +
BeachTypeToStr(Rec.BeachTypeName));
INC(index);

Rec.NumOfSpots := STRTOINT(SHRec.OwnedString[index]);
WriteLine("Number Of Spots is " + INTTOSTR(Rec.NumOfSpots));
INC(index);

NEW(Rec.SpotArray, 1..Rec.NumOfSpots);
OUTPUT("index = " + INTTOSTR(index));

IF (Rec.NumOfSpots > 0)
   i := 1;
   WHILE i <= Rec.NumOfSpots
      NEW(Spot);
      Spot.SpotClassification :=
StrToSpotType(SHRec.OwnedString[index]);

      WriteLine("Spot "+ INTTOSTR(i) + " is Type " +
                SpotTypeToStr(Spot.SpotClassification));

      OUTPUT("Spot "+ INTTOSTR(i) + " is Type " +
             SpotTypeToStr(Spot.SpotClassification));
```

```
      Char := SCHAR(SHRec.OwnedString[index+1], 1);
      CASE Char
        WHEN 'T','t' : Spot.SpotFree := TRUE;
        WHEN 'F','f' : Spot.SpotFree := FALSE;
      END CASE;
      WriteLine("Spot "+ INTTOSTR(i) + " is Free (T/F) " +
                BooleanToStr(Spot.SpotFree));
      OUTPUT("Spot "+ INTTOSTR(i) + " is Free (T/F) " +
             Char);

      Rec.SpotArray[i] := Spot;
      i := i + 1;
      index := index + 2;
   END WHILE;
END IF;

index := index;


Rec.DistanceFromSLCP := STRTOREAL(SHRec.OwnedString[index]);
WriteLine("DistanceFromSLCP is " +
REALTOSTR(Rec.DistanceFromSLCP));
INC(index);

Rec.DistBeachToArea := STRTOREAL(SHRec.OwnedString[index]);
WriteLine("DistBeachToArea is " +
REALTOSTR(Rec.DistBeachToArea));
INC(index);


OUTPUT("finished reading Beach type information");

RETURN(Rec);

END METHOD;

END OBJECT;


OBJECT BeachTypeListObj;
```

```
{-------------------------------------------------------------}
ASK METHOD ReadBeaches(IN BeachType : STRING);
{-------------------------------------------------------------}

VAR
Rec : BeachTypeRecType;

BEGIN
IF BeachTypeIOHandler = NILOBJ
   NEW(BeachTypeIOHandler);
   ASK BeachTypeIOHandler TO ReadRecs("BchType.dat");
END IF;

OUTPUT("Beach handler instanciated and full of raw
records");
OUTPUT("about to produce type record for " + BeachType);
Rec := ASK BeachTypeIOHandler TO ProduceRec(BeachType);
OUTPUT(" got the record complete ");
IF Rec <> NILREC
   Add(Rec);
ELSE
   OUTPUT(" never found record!");
END IF;

END METHOD;

END OBJECT;

END MODULE.



DEFINITION MODULE RFAName;

{-------------------------------------------------------------
 Module Name: RFAName                  Last Modified: 28 Jul 93
 Author:      J. S. Noel
              Lt.    USN
```

```
DESCRIPTION:  Defines the RefuelNameListObj and the
RefuelNameIORecHandleObj which together provide the means
for inputing the Refuel area Names into the simulation for a
given scenario.
-------------------------------------------------------------}
FROM RecIOHandle IMPORT RecIOHandleObj;
FROM ListMod     IMPORT QueueList;


TYPE


RefuelNameRecType = RECORD
   RefuelName          : STRING;

END RECORD;

RefuelNameListObj = OBJECT(QueueList[ANYREC :
                           RefuelNameRecType])
   ASK METHOD ReadRefuelNames;
END OBJECT;

RefuelNameIORecHandleObj = OBJECT(RecIOHandleObj[ANYREC :
                                  RefuelNameRecType])
END OBJECT;

VAR

RefuelNameIOHandler : RefuelNameIORecHandleObj;
MasterRefuelNameList : RefuelNameListObj;

END MODULE.
```

```
IMPLEMENTATION MODULE RFAName;

{-------------------------------------------------------------
 Module Name: RFAName                  Last Modified: 28 Jul 93
 Author:      J. S. Noel
              Lt.    USN

DESCRIPTION:  Implements the RefuelNameListObj and the
RefuelNameIORecHandleObj which together provide the means
for inputing the Refuel area Names into the simulation for a
given scenario.
-------------------------------------------------------------}
FROM RecIOHandle IMPORT SHArrayType;


OBJECT RefuelNameListObj;

{-------------------------------------------------------------}
ASK METHOD ReadRefuelNames;
{-------------------------------------------------------------}

VAR
Rec      : RefuelNameRecType;
index    : INTEGER;
high     : INTEGER;
SHArray  : SHArrayType;


BEGIN

IF RefuelNameIOHandler = NILOBJ
   NEW(RefuelNameIOHandler);
   ASK RefuelNameIOHandler TO ReadRecs("RFAName.dat");
END IF;

SHArray := ASK RefuelNameIOHandler SHArray;
high := HIGH(SHArray);

FOR index := 1 TO high
   NEW(Rec);
```

```
   Rec.RefuelName := SHArray[index].TopString;
   Add(Rec);

END FOR;

END METHOD;

END OBJECT;

END MODULE.


DEFINITION MODULE RFAList;

{-------------------------------------------------------------

Module Name: RFAList                    Last Modified: 28 Jul 93
Author:  J. S. Noel
         Lt.    USN

DESCRIPTION:  Defines the RefuelTypeListObj and the
RefuelTypeIORecHandleObj which together provide the means
for inputing the Refuel area into the simulation for a given
scenario.
-------------------------------------------------------------}
FROM RecIOHandle IMPORT RecIOHandleObj;
FROM ListMod     IMPORT QueueList;
FROM Global      IMPORT RefuelSpotRecType;
FROM Refuel      IMPORT RefuelSpotArrayType;

TYPE

RefuelTypeRecType = RECORD

   AreaTypeName          : STRING;
   NumOfSpots            : INTEGER;
   DistAreaToShip        : REAL;
   SpotArray             : RefuelSpotArrayType;
   FuelPumpRate          : REAL;
```

```
END RECORD;

RefuelTypeListObj = OBJECT(QueueList[ANYREC :
                           RefuelTypeRecType])
   ASK METHOD ReadRefuelArea(IN RefuelAreaName : STRING);
END OBJECT;

RefuelTypeIORecHandleObj = OBJECT(RecIOHandleObj[ANYREC :
                                  RefuelTypeRecType])
   OVERRIDE
   ASK METHOD ProduceRec(IN HeadString : STRING):
                         RefuelTypeRecType;
END OBJECT;

VAR

RefuelTypeIOHandler : RefuelTypeIORecHandleObj;
MasterRefuelTypeLis' : RefuelTypeListObj;

END MODULE.


IMPLEMENTATION MODULE RFAList;

{-------------------------------------------------------------
Module Name: RFAList                    Last Modified: 28 Jul 93
Author:  J. S. Noel
         Lt.    USN

DESCRIPTION:  Implements the RefuelTypeListObj and the
RefuelTypeIORecHandleObj which together provide the means
for inputing the Refuel area into the simulation for a given
scenario.
-------------------------------------------------------------}
FROM RecIOHandle IMPORT SHRecType;
FROM WriteLine   IMPORT WriteLine;
FROM Global      IMPORT RefuelSpotRecType;
FROM Convert     IMPORT BooleanToStr;

OBJECT RefuelTypeIORecHandleObj;
```

```
{-------------------------------------------------------------}
ASK METHOD ProduceRec(IN HeadString : STRING):
                        RefuelTypeRecType;
{-------------------------------------------------------------}

VAR
SHRec : SHRecType;
Rec : RefuelTypeRecType;
index, i : INTEGER;
Char : CHAR;
Spot : RefuelSpotRecType;


BEGIN
WriteLine("  producing record Refuel Area type " +
HeadString);
OUTPUT("  producing record Refuel Area type " + HeadString);
FindSHRec(HeadString, SHRec);
WriteLine(" ");
IF SHRec = NILREC
   OUTPUT("No record Found of " + HeadString);
   WriteLine("No record Found of " + HeadString);
   HALT;
END IF;
OUTPUT(" got SHRec");

NEW(Rec);
Rec.AreaTypeName := SHRec.TopString;
OUTPUT("AreaTypeName is " + Rec.AreaTypeName);
WriteLine("AreaTypeName is " + Rec.AreaTypeName);
index := 1;

Rec.NumOfSpots := STRTOINT(SHRec.OwnedString[index]);
WriteLine("Number Of Spots is " + INTTOSTR(Rec.NumOfSpots));
INC(index);

Rec.DistAreaToShip := STRTOREAL(SHRec.OwnedString[index]);
WriteLine("DistAreaToShip is " +
REALTOSTR(Rec.DistAreaToShip));
```

```
INC(index);

NEW(Rec.SpotArray, 1..Rec.NumOfSpots);
OUTPUT("index = " + INTTOSTR(index));

IF (Rec.NumOfSpots > 0)
   i := 1;
   WHILE i <= Rec.NumOfSpots
      NEW(Spot);

      Char := SCHAR(SHRec.OwnedString[index], 1);
      CASE Char
        WHEN 'T','t' : Spot.RefuelSpotFree := TRUE;
        WHEN 'F','f' : Spot.RefuelSpotFree := FALSE;
      END CASE;
      WriteLine("Spot "+ INTTOSTR(i) + " is Free (T/F) " +
                BooleanToStr(Spot.RefuelSpotFree));
      OUTPUT("Spot "+ INTTOSTR(i) + " is Free (T/F) " +
              Char);

      Rec.SpotArray[i] := Spot;
      i := i + 1;
      index := index + 1;
   END WHILE;
END IF;

index := index;

Rec.FuelPumpRate := STRTOREAL(SHRec.OwnedString[index]);
WriteLine("FuelPumpRate is " + REALTOSTR(Rec.FuelPumpRate));
INC(index);

OUTPUT("finished reading RefuelArea type information");

RETURN(Rec);

END METHOD;

END OBJECT;
```

```
OBJECT RefuelTypeListObj;

{-------------------------------------------------------------}
ASK METHOD ReadRefuelArea(IN RefuelAreaType : STRING);
{-------------------------------------------------------------}

VAR
Rec : RefuelTypeRecType;

BEGIN
IF RefuelTypeIOHandler = NILOBJ
   NEW(RefuelTypeIOHandler);
   ASK RefuelTypeIOHandler TO ReadRecs("RFAType.dat");
END IF;

OUTPUT("Beach handler instanciated and full of raw
records");
OUTPUT("about to produce type record for " +
RefuelAreaType);
Rec := ASK RefuelTypeIOHandler TO
ProduceRec(RefuelAreaType);
OUTPUT(" got the record complete ");
IF Rec <> NILREC
   Add(Rec);
ELSE
   OUTPUT(" never found record!");
END IF;

END METHOD;

END OBJECT;

END MODULE.
```

```
DEFINITION MODULE Builder;

{-------------------------------------------------------------
 Module Name: Builder                    Last Modified: 26 Jul 93
 Author:      J. S. Noel
               Lt.  USN

DESCRIPTION:  Defines the ObjectBuilderObj as well as the
four other Queue Objects required to build and store the
RoRo, Lighter, Beach, and FuelArea objects required for the
users scenario.
-------------------------------------------------------------}
FROM GrpMod      IMPORT QueueObj;
FROM Ship        IMPORT RoRoObj;
FROM Lighter     IMPORT LighterObj;
FROM Beach       IMPORT BeachObj;
FROM Refuel      IMPORT RefuelAreaObj;



TYPE

ObjectBuilderObj = OBJECT
   ASK METHOD BuildObjects;

END OBJECT;

ShipBuilderObj = OBJECT(QueueObj[ANYOBJ : RoRoObj])
   ASK METHOD BuildShips;

END OBJECT;

LighterBuilderObj = OBJECT(QueueObj[ANYOBJ : LighterObj])
   ASK METHOD BuildLighters;

END OBJECT;

BeachBuilderObj = OBJECT(QueueObj[ANYOBJ : BeachObj])
   ASK METHOD BuildBeaches;
```

```
END OBJECT;

FuelAreaBuilderObj = OBJECT(QueueObj[ANYOBJ :
RefuelAreaObj])
   ASK METHOD BuildFuelAreas;

END OBJECT;

VAR

ObjectBuilder   : ObjectBuilderObj;
ShipBuilder     : ShipBuilderObj;
LighterBuilder  : LighterBuilderObj;
BeachBuilder    : BeachBuilderObj;
FuelAreaBuilder : FuelAreaBuilderObj;

END MODULE.

IMPLEMENTATION MODULE Builder;


{-------------------------------------------------------------

 Module Name: Builder                  Last Modified: 26 Jul 93
 Author:      J. S. Noel
               Lt.  USN

DESCRIPTION:  Implements the ObjectBuilderObj as well as the
four other Queue Objects required to build and store the
RoRo,,Lighter, Beach, and FuelArea objects required for the
users scenario.
-------------------------------------------------------------}
FROM Ship      IMPORT RoRoObj;
FROM Lighter   IMPORT LighterObj;
FROM Beach     IMPORT BeachObj;
FROM Refuel    IMPORT RefuelAreaObj;
FROM ShpList   IMPORT MasterShipTypeList, ShipTypeRecType;
FROM LtList    IMPORT MasterLighterTypeList,
                      LighterTypeRecType;
FROM BchList   IMPORT MasterBeachTypeList, BeachTypeRecType;
```

```
FROM RFAList    IMPORT MasterRefuelTypeList,
                       RefuelTypeRecType;
FROM ShpName    IMPORT MasterShipNameList;
FROM LtName     IMPORT MasterLighterNameList;
FROM BchName    IMPORT MasterBeachNameList;
FROM RFAName    IMPORT MasterRefuelNameList;
FROM ShpName    IMPORT ShipNameRecType;
FROM LtName     IMPORT LighterNameRecType;
FROM BchName    IMPORT BeachNameRecType;
FROM RFAName    IMPORT RefuelNameRecType;
FROM WriteLine  IMPORT WriteLine;
FROM SLCP       IMPORT WaitForShipQueue;
FROM Convert    IMPORT LighterNameTypeToStr;
FROM RepMngr    IMPORT RepManager;



OBJECT ObjectBuilderObj;


{-------------------------------------------------------------}
ASK METHOD BuildObjects;
{-------------------------------------------------------------}

BEGIN


NEW(ShipBuilder);
   ASK ShipBuilder TO BuildShips;

NEW(LighterBuilder);
   ASK LighterBuilder TO BuildLighters;

NEW(BeachBuilder);
   ASK BeachBuilder TO BuildBeaches;

NEW(FuelAreaBuilder);
   ASK FuelAreaBuilder TO BuildFuelAreas;
```

```
END METHOD;

END OBJECT;

{============================================================}

OBJECT ShipBuilderObj;

{-----  ------------------------------------------------------}
ASK METHOD BuildShips;
{------------------------------------------------------------}

VAR

Rec : ShipTypeRecType;
RoRo : RoRoObj;

BEGIN


NEW(Rec);
NEW(RoRo);

Rec := ASK MasterShipTypeList First();

{
WriteLine("Building ship " + Rec.ShipTypeName);
WriteLine("  ");
}

WHILE Rec <> NILREC
   ASK RoRo TO GetShipSetup(Rec.ShipTypeName, Rec.TypeShip,
                            Rec.DistanceFromBLCP,
                            Rec.NumOfSpots, Rec.SpotArray,
                            Rec.NumOfLoLoVehicles,
                            Rec.NumOfRRDFVehicles,
                            Rec.NumOfAnySpotVehicles);
   Add(RoRo);
```

```
NEW(RoRo);
Rec := ASK MasterShipTypeList Next(Rec);
END WHILE;

END METHOD;


END OBJECT;

{=============================================================}

OBJECT LighterBuilderObj;

{-------------------------------------------------------------}
ASK METHOD BuildLighters;
{-------------------------------------------------------------}

VAR

Rec : LighterTypeRecType;
Lighter : LighterObj;

BEGIN
{
WriteLine("Building Lighter ");
WriteLine("  ");
}

NEW(Rec);
NEW(WaitForShipQueue);
NEW(Lighter);

Rec := ASK MasterLighterTypeList First();
WHILE Rec <> NILREC
   ASK Lighter TO GetLighterSetup(Rec.ID,
                        Rec.LighterTypeName,
                        Rec.SpotRequired, Rec.SpeedMax,
                        Rec.SpeedFull, Rec.MaxLoad,
                        Rec.FuelCap, Rec.BurnRate,
                        Rec.MinFuel);
```

```
   Add(Lighter);
   ASK WaitForShipQueue TO Add(Lighter);

{
   WriteLine("Number in ship Q is " +
             INTTOSTR(ASK WaitForShipQueue numberIn) + "
             Lighter Name= " +
             LighterNameTypeToStr(ASK Lighter
             LighterTypeName)
             + " LighterID = " + ASK Lighter LighterID);
}

NEW(Lighter);
Rec := ASK MasterLighterTypeList Next(Rec);
END WHILE;


END METHOD;

END OBJECT;

{=========================================================}

OBJECT BeachBuilderObj;

{---------------------------------------------------------}
ASK METHOD BuildBeaches;
{---------------------------------------------------------}

VAR

Rec : BeachTypeRecType;
Beach : BeachObj;

BEGIN
{
WriteLine("Building Beach ");
WriteLine("  ");
}
```

```
NEW(Rec);
NEW(Beach);

Rec := ASK MasterBeachTypeList First();
WHILE Rec <> NILREC
   ASK Beach TO GetBeachSetup(Rec.ID, Rec.BeachTypeName,
                        Rec.NumOfSpots, Rec.SpotArray,
                        Rec.DistanceFromSLCP,
                        Rec.DistBeachToArea);
   Add(Beach);


NEW(Beach);
Rec := ASK MasterBeachTypeList Next(Rec);
END WHILE;

END METHOD;


END OBJECT;

{=========================================================}

OBJECT FuelAreaBuilderObj;

{---------------------------------------------------------}
ASK METHOD BuildFuelAreas;
{---------------------------------------------------------}

VAR

Rec : RefuelTypeRecType;
RefuelArea : RefuelAreaObj;

BEGIN
{
WriteLine("Building RefuelArea ");
WriteLine("  ");
}
```

```
NEW(Rec);
NEW(RefuelArea);

Rec := ASK MasterRefuelTypeList First();
WHILE Rec <> NILREC
   ASK RefuelArea TO GetRefuelAreaSetup(Rec.AreaTypeName,
                        Rec.NumOfSpots,
                         Rec.DistAreaToShip,
                         Rec.SpotArray, Rec.FuelPumpRate);
                         Add(RefuelArea);


NEW(RefuelArea);
Rec := ASK MasterRefuelTypeList Next(Rec);
END WHILE;

END METHOD;
END OBJECT;

END MODULE.



DEFINITION MODULE RepMngr;

{-------------------------------------------------------------
Module Name: RepMngr                    Last Modified: 26 Jul 93
Author:  M. Bailey                      Modified By: J. S. Noel
         Prof.  NPG                                  Lt.  USN

DESCRIPTION:  Defines the replication manager  ( RepMngrObj)
for initialization and operation of the simulation.
-------------------------------------------------------------}

TYPE

RepMngrObj = OBJECT

   MaxNumberOfReps : INTEGER;
```

```
   Iteration        : INTEGER;
   Done             : BOOLEAN;
   OutputToScreen   : BOOLEAN;
   SeedAlfa         : INTEGER;
   SeedBravo        : INTEGER;

   ASK METHOD ObjInit;
   ASK METHOD ChangeRunParms;
   ASK METHOD PrepForRep;
   ASK METHOD Replicate;
   ASK METHOD ResetForNextRun;

END OBJECT;

VAR

RepManager : RepMngrObj;

END MODULE.



IMPLEMENTATION MODULE RepMngr;

{-------------------------------------------------------------
Module Name: RepMngr                  Last Modified: 26 Jul 93
Author:   M. Bailey                   Modified By: J. S. Noel
          Prof.  NPG                                 Lt.  USN

DESCRIPTION:  Implements the replication manager
( RepMngrObj) for initialization and operation of the
simulation.
-------------------------------------------------------------}
FROM WriteLine IMPORT WriteLine;
FROM SimMod    IMPORT StartSimulation, ResetSimTime;
FROM CRTMod    IMPORT ClearScreen;
FROM IOMod     IMPORT ReadKey;
FROM Ship      IMPORT RoRoObj;
FROM SLCP      IMPORT SLCP;
FROM BLCP      IMPORT BLCP;
```

```
FROM FuelCP     IMPORT FuelCP;
FROM Builder    IMPORT ShipBuilder;
FROM Stats      IMPORT Stats;
FROM Global     IMPORT RandTime1,RandTime2;

OBJECT RepMngrObj;

{-------------------------------------------------------------}
ASK METHOD ObjInit;
{-------------------------------------------------------------}

BEGIN

MaxNumberOfReps := 1;
Iteration := 0;
Done := FALSE;
OutputToScreen := FALSE;
SeedAlfa := 123456;
SeedBravo := 678912;


END METHOD;


{-------------------------------------------------------------}
ASK METHOD ChangeRunParms;
{-------------------------------------------------------------}

VAR

Ch : CHAR;
Seed1 : INTEGER;
Seed2 : INTEGER;

BEGIN

ClearScreen;
OUTPUT;OUTPUT;
OUTPUT("??????????????????????????????????????????????????");
OUTPUT;OUTPUT;
```

```
OUTPUT("The number of replications desired is ...");
OUTPUT;OUTPUT;
OUTPUT("??????????????????????????????????????????????????");
OUTPUT;OUTPUT;
INPUT(MaxNumberOfReps);


ClearScreen;
OUTPUT;OUTPUT;
OUTPUT("??????????????????????????????????????????????????");
OUTPUT;OUTPUT;
OUTPUT("Do you want output displayed on the screen?");
OUTPUT;OUTPUT;
OUTPUT("??????????????????????????????????????????????????");
OUTPUT;OUTPUT;
Ch := ReadKey();
IF (Ch = 'y') OR (Ch = 'Y') OR (Ch = 't') OR (Ch = 'T')
   OutputToScreen := TRUE;
END IF;

ClearScreen;
OUTPUT;OUTPUT;
OUTPUT("??????????????????????????????????????????????????");
OUTPUT;OUTPUT;
OUTPUT("Do you want to input seeds?");
OUTPUT;OUTPUT;
OUTPUT("??????????????????????????????????????????????????");
OUTPUT;OUTPUT;
Ch := ReadKey();
IF (Ch = 'y') OR (Ch = 'Y') OR (Ch = 't') OR (Ch = 'T')
   OUTPUT("Input seed number 1.  MUST BE INTEGER ");
   INPUT(SeedAlfa);
   OUTPUT("Input seed number 2.  MUST BE INTEGER ");
   INPUT(SeedBravo);

END IF;

NEW(RandTime1);
NEW(RandTime2);
```

```
ASK RandTime1 TO SetSeed(SeedAlfa);
ASK RandTime2 TO SetSeed(SeedBravo);

END METHOD;


{-------------------------------------------------------------}
ASK METHOD Replicate;
{-------------------------------------------------------------}

VAR

RoRo : RoRoObj;

BEGIN

FOR Iteration := 1 TO MaxNumberOfReps
   ResetSimTime(0.0);
   WriteLine("______replication " + INTTOSTR(Iteration) +
             "______");
   ASK SELF TO PrepForRep;
   ASK SELF TO ResetForNextRun;

   IF (Iteration = MaxNumberOfReps)
      Done := TRUE;
      WriteLine("//RepManager//  Done = TRUE");
   END IF;

   OUTPUT("Simulation clock started.");
   WriteLine("______CLOCK STARTS FOR REP " +
             INTTOSTR(Iteration) + "______");
   StartSimulation;


   WriteLine("______CLOCK STOPS FOR REP " +
             INTTOSTR(Iteration) + "______");

END FOR;

END METHOD;
```

```
{-------------------------------------------------------------}
ASK METHOD PrepForRep;
{-------------------------------------------------------------}

BEGIN

IF ( Iteration = 1 )
   NEW(SLCP);
   NEW(BLCP);
   NEW(FuelCP);
   NEW(Stats);
END IF;


END METHOD;


{-------------------------------------------------------------}
ASK METHOD ResetForNextRun;
{-------------------------------------------------------------}

VAR

RoRo : RoRoObj;

BEGIN

WriteLine("----Reset For Next Run----");


IF ( NOT Done )

   RoRo := ASK ShipBuilder First();
   WHILE RoRo <> NILOBJ
      ASK RoRo TO StartTheShow;

   RoRo := ASK ShipBuilder Next(RoRo);
   END WHILE;

END IF;
```

```
END METHOD;


END OBJECT;

END MODULE.


DEFINITION MODULE Global;

{-------------------------------------------------------------

Module Name: Global                    Last Modified: 18 Jun 93
Author:     J. S. Noel
            Lt.     USN

DESCRIPTION:  Defines Global types used in the simulation.
-------------------------------------------------------------}
FROM RandMod   IMPORT RandomObj;


TYPE

   SpotType = (LCU, CWF, LoLo);

   SpotRecType = RECORD
      SpotClassification : SpotType;
      SpotFree           : BOOLEAN;
      TotalIdleTime      : REAL;
   END RECORD;

   RefuelSpotRecType = RECORD
      RefuelSpotFree     : BOOLEAN;
   END RECORD;

   SpotIdleTimeRecType = RECORD
      StartTime : REAL;
      EndTime   : REAL;
   END RECORD;
```

```
   NameRecType = RECORD
      Name : STRING;
   END RECORD;

   SpotIdleTimeArrayType = ARRAY INTEGER OF
                           SpotIdleTimeRecType;

   FileNameType = STRING;

   DestinationType = (Ship, Bch, Fuel, RefuelFromBeach,
                      ShipFromRefuel);

   SeedArrayType = ARRAY INTEGER OF INTEGER;

VAR

RandTime1 : RandomObj;
RandTime2 : RandomObj;

END MODULE.


IMPLEMENTATION MODULE Global;

{-------------------------------------------------------------

Module Name: Global                    Last Modified: 18 Jun 93
Author:   J. S. Noel
          Lt.    USN

DESCRIPTION: Implements Global types used in the simulation.
-------------------------------------------------------------}

{ The functioning components of Global ar in the Definition
Module.}

END MODULE.
```

```
DEFINITION MODULE Ship;

{-------------------------------------------------------------
Module Name: Ship                          Last Modified: 18 Jun 93
Author:  J. S. Noel
         Lt.    USN

DESCRIPTION:  Defines a RoRo (Ship) object.
-------------------------------------------------------------}
FROM ShpList IMPORT SpotArrayType;
FROM Global  IMPORT SpotIdleTimeArrayType, SpotType;
FROM Lighter IMPORT LighterObj;

TYPE

ShipTypeType = (SSR, NSSR);

RoRoObj = OBJECT

   ShipName            : STRING;
   ShipType            : ShipTypeType;
   DistanceFromBLCP    : REAL;
   NumSpots            : INTEGER;
   ShipSpot            : SpotArrayType;
   NumLoLoVehicles     : INTEGER;
   NumRRDFVehicles     : INTEGER;
   NumAnySpotVehicles  : INTEGER;
   PermNumLoLo         : INTEGER;
   PermNumRRDF         : INTEGER;
   PermNumAnySpot      : INTEGER;

   LoadSize            : INTEGER;
   ShipSpotIdleTime    : SpotIdleTimeArrayType;
   LastLoad            : BOOLEAN;

   ASK METHOD ObjInit;
   ASK METHOD StartTheShow;
   ASK METHOD GetShipSetup(IN Name : STRING;
                           IN Type : ShipTypeType;
                           IN Num1 : REAL;
```

```
                                IN Num2 : INTEGER;
                                IN Array : SpotArrayType;
                                IN Num3 : INTEGER;
                                IN Num4 : INTEGER;
                                IN Num5 : INTEGER);

   ASK METHOD LogIdleShipSpotTime(IN index : INTEGER;
                                  IN InOutSpotTime : REAL;
                                  IN SpotIdle : BOOLEAN);
   ASK METHOD MakeLoad(IN Lighter : LighterObj;
                       IN index : INTEGER);
   ASK METHOD SetSpotFree(IN index : INTEGER);
   ASK METHOD CheckSpots(IN SpotTyp : SpotType;
                         OUT SpotAvail : BOOLEAN;
                         OUT index : INTEGER);
   ASK METHOD OccupySpot(IN index : INTEGER);
   ASK METHOD ResetShipStats;


END OBJECT;

VAR
RoRo : RoRoObj;

END MODULE.


IMPLEMENTATION MODULE Ship;

{-------------------------------------------------------------
Module Name: Ship                      Last Modified: 18 Jun 93
Author:  J. S. Noel
         Lt.    USN

DESCRIPTION:  Implements a RoRo (Ship) object.
-------------------------------------------------------------}
FROM ShpList    IMPORT SpotArrayType;
FROM Global     IMPORT ALL SpotType, SpotRecType,
                       ALL SpotIdleTimeRecType;
FROM SimMod     IMPORT SimTime;
```

```
FROM SLCP       IMPORT SLCP;
FROM Lighter    IMPORT LighterObj;
FROM WriteLine IMPORT WriteLine;
FROM Convert    IMPORT BooleanToStr, SpotTypeToStr;


OBJECT RoRoObj;

{-------------------------------------------------------------}
ASK METHOD ObjInit;
{-------------------------------------------------------------}

BEGIN

LastLoad := FALSE;

END METHOD;

{-------------------------------------------------------------}
ASK METHOD StartTheShow;
{-------------------------------------------------------------}

VAR
i : INTEGER;

BEGIN

FOR i := 1 TO HIGH(ShipSpot)

{
   WriteLine("SpotFree for spot " + INTTOSTR(i) + " is " +
             BooleanToStr(ShipSpot[i].SpotFree));
}

   ASK SLCP TO GetLighter(ShipSpot[i].SpotClassification, i,
SELF);
END FOR;


END METHOD;
```

```
{-------------------------------------------------------------}
ASK METHOD GetShipSetup(IN Name : STRING;
                        IN Type : ShipTypeType;
                        IN Num1 : REAL;
                        IN Num2 : INTEGER;
                        IN Array : SpotArrayType;
                        IN Num3 : INTEGER;
                        IN Num4 : INTEGER;
                        IN Num5 : INTEGER);
{-------------------------------------------------------------}

VAR
i : INTEGER;
Rec : SpotIdleTimeRecType;

BEGIN

ShipName := Name;

{
WriteLine("//GetShipSetup//  ShipName = " + ShipName);
}

ShipType := Type;
DistanceFromBLCP :. Num1;
NumSpots := Num2;

NEW(ShipSpot, 1..NumSpots);
ShipSpot := Array;

{
WriteLine(ShipName + " Spot " + INTTOSTR(1) + " SpotType " +
          SpotTypeToStr(ShipSpot[1].SpotClassification));
}

NumLoLoVehicles := Num3;
NumRRDFVehicles := Num4;
NumAnySpotVehicles := Num5;
```

```
NEW(ShipSpotIdleTime, 1..NumSpots);

FOR i := 1 TO NumSpots
   ShipSpot[i].TotalIdleTime := 0.0;

   NEW(Rec);
   Rec.StartTime := 0.0;
   Rec.EndTime := 0.0;
   ShipSpotIdleTime[i] := Rec;

END FOR;

PermNumLoLo := NumLoLoVehicles;
PermNumRRDF := NumRRDFVehicles;
PermNumAnySpot := NumAnySpotVehicles;


END METHOD;

{-------------------------------------------------------------}
ASK METHOD LogIdleShipSpotTime(IN index : INTEGER;
                               IN InOutSpotTime : REAL;
                               IN SpotIdle : BOOLEAN);
{-------------------------------------------------------------}

BEGIN

{
WriteLine("LogIdleSpotTime fired on " + ShipName);
}

IF SpotIdle
   ShipSpotIdleTime[index].StartTime := InOutSpotTime;

{
   WriteLine("Spot " + INTTOSTR(index) + " Idle at " +
             REALTOSTR(InOutSpotTime));
   WriteLine("  ");
}
```

```
ELSE
   ShipSpotIdleTime[index].EndTime := InOutSpotTime;

{
   WriteLine("Spot " + INTTOSTR(index) + " Occupied at " +
                  REALTOSTR(InOutSpotTime));
   WriteLine("   ");
}

   ShipSpot[index].TotalIdleTime :=
                        ShipSpot[index].TotalIdleTime +
                       (ShipSpotIdleTime[index].EndTime -
                        ShipSpotIdleTime[index].StartTime);

END IF;

END METHOD;


{-------------------------------------------------------------}
ASK METHOD MakeLoad(IN Lighter : LighterObj;
                    IN index : INTEGER);
{-------------------------------------------------------------}

VAR
LoadSize : INTEGER;

BEGIN

{
WriteLine("MakeLoad fired for spot " + INTTOSTR(index) +
          " on " + ShipName );
}

ASK SELF TO OccupySpot(index);
LoadSize := ASK Lighter MyLoadSize;

IF ( ShipSpot[index].SpotClassification = LoLo )
   IF ( NumLoLoVehicles > 0 )
```

```
   IF ( NumLoLoVehicles < LoadSize )
      LoadSize := LoadSize - NumLoLoVehicles;
      NumLoLoVehicles := 0;

      IF ( NumAnySpotVehicles > 0 )
         IF ( NumAnySpotVehicles < LoadSize )
            NumAnySpotVehicles := 0;
            LastLoad := TRUE;
            ASK SLCP TO SetShipStatus;
            ASK Lighter TO SetLoadStatus(LastLoad, SELF);
         ELSE { NumAnySpotVehicles > LoadSize }
            NumAnySpotVehicles := NumAnySpotVehicles -
                                  LoadSize;
         END IF;

      ELSE { NumAnySpotVehicles = 0 }
         LastLoad := TRUE;
         ASK SLCP TO SetShipStatus;
         ASK Lighter TO SetLoadStatus(LastLoad, SELF);
      END IF;

   ELSE { NumLoLoVehicles > LoadSize }
      NumLoLoVehicles := NumLoLoVehicles - LoadSize;

   END IF;

ELSE { NumLoLoVehicles = 0 }

   IF ( NumAnySpotVehicles > 0 )
      IF ( NumAnySpotVehicles < LoadSize )
         NumAnySpotVehicles := 0;
         LastLoad := TRUE;
         ASK SLCP TO SetShipStatus;
         ASK Lighter TO SetLoadStatus(LastLoad, SELF);
      ELSE { NumAnySpotVehicles > LoadSize }
            NumAnySpotVehicles := NumAnySpotVehicles -
                                  LoadSize;
      END IF;

   ELSE { NumAnySpotVehicles = 0 }
```

```
            LastLoad := TRUE;
            ASK SLCP TO SetShipStatus;
            ASK Lighter TO SetLoadStatus(LastLoad, SELF);
         END IF;
      END IF;

   ELSE { ShipSpot[index].SpotClassification = LCU or CWF }

      IF ( NumRRDFVehicles > 0 )
         IF ( NumRRDFVehicles < LoadSize )
            LoadSize := LoadSize - NumRRDFVehicles;
            NumRRDFVehicles := 0;

            IF ( NumAnySpotVehicles > 0 )
               IF ( NumAnySpotVehicles < LoadSize )
                  NumAnySpotVehicles := 0;
                  LastLoad := TRUE;
                  ASK SLCP TO SetShipStatus;
                  ASK Lighter TO SetLoadStatus(LastLoad, SELF);
               ELSE { NumAnySpotVehicles > LoadSize }
                  NumAnySpotVehicles := NumAnySpotVehicles -
                                        LoadSize;
               END IF;

            ELSE { NumAnySpotVehicles = 0 }
               LastLoad := TRUE;
               ASK SLCP TO SetShipStatus;
               ASK Lighter TO SetLoadStatus(LastLoad, SELF);
            END IF;

         ELSE
            NumRRDFVehicles := NumRRDFVehicles - LoadSize;

         END IF;

      ELSE { NumRRDFVehicles = 0 }

         IF ( NumAnySpotVehicles > 0 )
            IF ( NumAnySpotVehicles < LoadSize )
               NumAnySpotVehicles := 0;
```

```
            LastLoad := TRUE;
            ASK SLCP TO SetShipStatus;
            ASK Lighter TO SetLoadStatus(LastLoad, SELF);
         ELSE { NumAnySpotVehicles > LoadSize }
               NumAnySpotVehicles := NumAnySpotVehicles -
                                     LoadSize;
         END IF;

      ELSE { NumAnySpotVehicles = 0 }
         LastLoad := TRUE;
         ASK SLCP TO SetShipStatus;
         ASK Lighter TO SetLoadStatus(LastLoad, SELF);
      END IF;
   END IF;
END IF;

{
WriteLine("NumLoLoVehicles = " + INTTOSTR(NumLoLoVehicles));
WriteLine("NumRRDFVehicles = " + INTTOSTR(NumRRDFVehicles));
WriteLine("NumAnySpotVehicles = " +
          INTTOSTR(NumAnySpotVehicles));
}

TELL Lighter TO OnLoad(SELF);

END METHOD;

{-------------------------------------------------------------}
ASK METHOD SetSpotFree(IN index : INTEGER);
{-------------------------------------------------------------}

VAR
Idle : BOOLEAN;

BEGIN

{
WriteLine("SetSpotFree fired for spot " + INTTOSTR(index) +
          " on " +  ShipName );
}
```

```
IF ( NOT LastLoad )
   ShipSpot[index].SpotFree := TRUE;
   Idle := ShipSpot[index].SpotFree;
   ASK SLCP TOGetLighter(ShipSpot[index].SpotClassification,
                         index, SELF);
END IF;

ASK SELF TO LogIdleShipSpotTime(index, SimTime(), Idle);

END METHOD;

{-------------------------------------------------------------}
ASK METHOD CheckSpots(IN SpotTyp : SpotType;
                      OUT SpotAvail : BOOLEAN;
                      OUT index : INTEGER);
{-------------------------------------------------------------}
VAR

i          : INTEGER;

BEGIN

{
WriteLine("CheckSpots Fired on " + ShipName);
}

SpotAvail := FALSE;

FOR i := 1 TO HIGH(ShipSpot)
   IF (ShipSpot[i].SpotClassification = SpotTyp) AND
      (ShipSpot[i].SpotFree)

      SpotAvail := TRUE;
      index := i;
      ShipSpot[i].SpotFree := FALSE;
      EXIT;
   END IF;
END FOR;
```

```
END METHOD;


{-------------------------------------------------------------}
ASK METHOD OccupySpot(IN index : INTEGER);
{-------------------------------------------------------------}

VAR
Idle : BOOLEAN;

BEGIN

{
WriteLine("Occupy spot fired in RoRo " + ShipName);
}


ShipSpot[index].SpotFree := FALSE;
Idle := ShipSpot[index].SpotFree;

ASK SELF TO LogIdleShipSpotTime(index, SimTime(), Idle);

END METHOD;

{-------------------------------------------------------------}
ASK METHOD ResetShipStats;
{-------------------------------------------------------------}

VAR
i : INTEGER;

BEGIN

{
WriteLine("ResetShipStats fired on " + ShipName);
}

FOR i := 1 TO HIGH(ShipSpot)
   ShipSpot[i].SpotFree := TRUE;
```

```
   ShipSpot[i].TotalIdleTime := 0.0;
   ShipSpotIdleTime[i].StartTime := 0.0;
   ShipSpotIdleTime[i].EndTime := 0.0;
END FOR;

NumLoLoVehicles := PermNumLoLo;
NumRRDFVehicles := PermNumRRDF;
NumAnySpotVehicles := PermNumAnySpot;

LastLoad := FALSE;

END METHOD;


END OBJECT;

END MODULE.


DEFINITION MODULE SLCP;

{-------------------------------------------------------------
Module Name: SLCP                    Last Modified: 18 Jun 93
Author:    J. S. Noel
           Lt.     USN

DESCRIPTION:  Defines the Ships Lighterage Control Point
Object (SLCPObj).   After a Lighter CastsandClears the ship
it asks the RoRoObj to SetSpotFree.  This method fires the
GetLighter method in SLCPObj.  SLCPObj then pops the first
appropriate lighter off of theAwaitingShipQueue and directs
the lighter to  ApproachAndMoor to the RRDF, where the
onload of vehicles can begin.
-------------------------------------------------------------}
FROM GrpMod  IMPORT QueueObj;
FROM Global  IMPORT SpotType;
FROM Lighter IMPORT LighterObj;
FROM Ship    IMPORT RoRoObj;

TYPE
```

```
ShipDoneRecType = RECORD
   ShipDone : BOOLEAN;
END RECORD;

ShipDoneArrayType = ARRAY INTEGER OF ShipDoneRecType;

AwaitingShipQueueObj = OBJECT(QueueObj[ANYOBJ: LighterObj])
END OBJECT;

SLCPObj = OBJECT

   NumLighter       : INTEGER;
   NumShip          : INTEGER;
   ShipStatusIndex  : INTEGER;
   ShipsDone        : ShipDoneArrayType;
   AllDone          : BOOLEAN;

   ASK METHOD ObjInit;
   ASK METHOD SetShipStatus;
   ASK METHOD GetSpot(IN Lighter : LighterObj);
   ASK METHOD GetLighter(IN Spot : SpotType;
                         IN index : INTEGER;
                         IN RoRo : RoRoObj);
   ASK METHOD ResetSLCP;

END OBJECT;

VAR
WaitForShipQueue : AwaitingShipQueueObj;
SLCP : SLCPObj;

END MODULE.
```

```
IMPLEMENTATION MODULE SLCP;

{-------------------------------------------------------------
Module Name: SLCP                          Last Modified: 18 Jun 93
Author:  J. S. Noel
         Lt.    USN

DESCRIPTION:  Implements a Ship Lighter Control Point (SLCP)
object.
-------------------------------------------------------------}
FROM Global    IMPORT ALL SpotType, ALL DestinationType;
FROM Lighter   IMPORT LighterObj;
FROM Ship      IMPORT RoRoObj;
FROM SimMod    IMPORT SimTime;
FROM Builder   IMPORT ShipBuilder;
FROM WriteLine IMPORT WriteLine;
FROM Convert   IMPORT LighterNameTypeToStr;
FROM Stats     IMPORT Stats;
FROM Builder   IMPORT ShipBuilder, LighterBuilder;
FROM BLCP      IMPORT WaitForBeachQueue;


OBJECT SLCPObj;

{-------------------------------------------------------------}
ASK METHOD ObjInit;
{-------------------------------------------------------------}
VAR

i   : INTEGER;
Rec : ShipDoneRecType;

BEGIN

AllDone := FALSE;
ShipStatusIndex := 1;
NumLighter := ASK LighterBuilder numberIn;
NumShip := ASK ShipBuilder numberIn;
```

```
NEW(Rec);
NEW(ShipsDone, 1..NumShip);
FOR i := 1 TO HIGH(ShipsDone)
   NEW(Rec);
   Rec.ShipDone := FALSE;
   ShipsDone[i] := Rec;
END FOR;

END METHOD;


{-------------------------------------------------------------}
ASK METHOD SetShipStatus;
{-------------------------------------------------------------}
VAR

i : INTEGER;
RoRo : RoRoObj;

BEGIN

ShipsDone[ShipStatusIndex].ShipDone := TRUE;

IF ( ShipStatusIndex = NumShip )
   AllDone := TRUE;

   IF ( AllDone ) AND ( ASK WaitForShipQueue numberIn =
                        NumLighter )
   WriteLine("---Last Lighter in the Q, Dumping
             Stats.----");

   RoRo := ASK ShipBuilder First();
   WHILE RoRo <> NILOBJ
      FOR i := 1 TO (ASK RoRo NumSpots)

         ASK RoRo TO LogIdleShipSpotTime(i,SimTime(),FALSE);

      END FOR;

   RoRo := ASK ShipBuilder Next(RoRo);
```

```
   END WHILE;


   ASK Stats TO DumpStats(SimTime());

ELSE
   INC(ShipStatusIndex);

END IF;

END METHOD;


{-------------------------------------------------------------}
ASK METHOD GetSpot(IN Lighter : LighterObj);
{-------------------------------------------------------------}

VAR

SpotReq    : SpotType;
i          : INTEGER;
index      : INTEGER;
SpotAvail  : BOOLEAN;
LogIn      : BOOLEAN;
QType      : BOOLEAN;
RoRo       : RoRoObj;
Dest       : DestinationType;

BEGIN

{
WriteLine("GetSpot Fired in SLCP ");
}

SpotAvail := FALSE;
SpotReq := ASK Lighter LighterSpot;
Dest := Ship;

RoRo := ASK ShipBuilder First();
WHILE RoRo <> NILOBJ
```

```
   IF ( ASK RoRo LastLoad )
      RoRo := ASK ShipBuilder Next(RoRo);

   ELSE
      ASK RoRo TO CheckSpots(SpotReq, SpotAvail, i);

      IF SpotAvail
         TELL Lighter TO ApproachAndMoor(i, Dest, RoRo);
         EXIT;

      ELSE
         RoRo := ASK ShipBuilder Next(RoRo);

      END IF;

   END IF;

END WHILE;

IF ( NOT SpotAvail )
   ASK WaitForShipQueue TO Add(Lighter);
   LogIn := TRUE; { adding to Q }
   QType := TRUE; { Q type = ship }
   ASK Lighter TO LogQueueTime(SimTime(), LogIn, QType);

END IF;



IF ( AllDone ) AND ( ASK WaitForShipQueue numberIn =
                     NumLighter )
   WriteLine("---Last Lighter in the Q, Dumping
             Stats.----");

   RoRo := ASK ShipBuilder First();
   WHILE RoRo <> NILOBJ
      FOR i := 1 TO (ASK RoRo NumSpots)

         ASK RoRo TO LogIdleShipSpotTime(i,SimTime(),FALSE);
```

```
      END FOR;

   RoRo := ASK ShipBuilder Next(RoRo);
   END WHILE;


   ASK Stats TO DumpStats(SimTime());

END IF;


END METHOD;

{-------------------------------------------------------------}
ASK METHOD GetLighter(IN Spot : SpotType;
                      IN index : INTEGER;
                      IN RoRo : RoRoObj);
{-------------------------------------------------------------}

VAR

Lighter   : LighterObj;
LogIn     : BOOLEAN;
QType     : BOOLEAN;
Dest      : DestinationType;

BEGIN

{
WriteLine("GetLighter Fired in SLCP. Index = " +
          INTTOSTR(index));
}

Dest := Ship;

{
WriteLine("Number in ship Q At SLCP is " +
          INTTOSTR(ASK WaitForShipQueue numberIn));
}
```

```
Lighter := ASK WaitForShipQueue First();

{If Spot available is a LoLo spot then take the first
available from  the queue, ELSE, the lighter SpotType must
match the SpotType available for mooring to the RRDF.}

IF ( Spot = LoLo )

   IF Lighter <> NILOBJ
      ASK WaitForShipQueue TO RemoveThis(Lighter);
         LogIn := FALSE; { Not entering ship Q }
         QType := TRUE;  { Q type = ship }
         ASK Lighter TO LogQueueTime(SimTime(),LogIn,QType);
         TELL Lighter TO ApproachAndMoor(index, Dest, RoRo);

   END IF;

ELSE
   WHILE Lighter <> NILOBJ

      IF ( ASK Lighter LighterSpot = Spot )
         ASK WaitForShipQueue TO RemoveThis(Lighter);
         LogIn := FALSE; { Not entering ship Q }
         QType := TRUE;  { Q type = ship }
         ASK Lighter TO LogQueueTime(SimTime(),LogIn,QType);
         TELL Lighter TO ApproachAndMoor(index, Dest, RoRo);
      EXIT;
      END IF;

   Lighter := ASK WaitForShipQueue Next(Lighter);

   END WHILE;
END IF;



END METHOD;
```

```
{-------------------------------------------------------------}
ASK METHOD ResetSLCP;
{-------------------------------------------------------------}
VAR

i    : INTEGER;

BEGIN

AllDone := FALSE;
ShipStatusIndex := 1;

FOR i := 1 TO HIGH(ShipsDone)
   ShipDone[i].ShipDone := FALSE;

END FOR;

END METHOD;

END OBJECT;

END MODULE.



DEFINITION MODULE Lighter;

{-------------------------------------------------------------
Module Name: Lighter                    Last Modified: 17 Jul 93
Author:  J. S. Noel
         Lt.    USN

DESCRIPTION:  Defines a Lighter (Smallcraft or boat) object.
-------------------------------------------------------------}
FROM Global  IMPORT SpotType, DestinationType;
FROM Ship    IMPORT RoRoObj;
```

```
TYPE

LighterNameType = (LCU1466 ,LCU1610, LCU2000, CWF11, CWF21,
                   CWF31, LSV);

LighterObj = OBJECT

   LighterTypeName     : LighterNameType;
   LighterID           : STRING;
   LighterSpot         : SpotType;
   MaxSpeed            : REAL;
   FullLoadSpeed       : REAL;
   MyLoadSize          : INTEGER;
   ShipID              : RoRoObj;

   FuelCapacity        : REAL;
   CurrentFuel         : REAL;
   FuelBurnRate        : REAL;
   MinFuel             : REAL;
   MinFuelPercent      : REAL;

   TransitDistance     : REAL;
   TimeInShipQueue     : REAL;
   TimeInBeachQueue    : REAL;
   TimeToTotalOffload  : REAL;
   LoadStatus          : BOOLEAN;
   ShipSpotIndex       : INTEGER;
   BeachSpotIndex      : INTEGER;
   RefuelSpotIndex     : INTEGER;

   InShipQTime         : REAL;
   OutShipQTime        : REAL;
   InBeachQTime        : REAL;
   OutBeachQTime       : REAL;
```

```
    ASK METHOD ObjInit;
    ASK METHOD GetLighterSetup(IN ID : STRING;
                               IN Name : LighterNameType;
                               IN Sp : SpotType;
                               IN Num1 : REAL;
                               IN Num2 : REAL;
                               IN Num3 : INTEGER;
                               IN Num4 : REAL;
                               IN Num5 : REAL;
                               IN Num6 : REAL);

    ASK METHOD LogQueueTime(IN InOutQTime : REAL;
                            IN EnterQ : BOOLEAN;
                            IN ShipQ : BOOLEAN);
    ASK METHOD SetLoadStatus(IN Status : BOOLEAN;
                             IN Vessel : RoRoObj);
    ASK METHOD BurnFuel(IN BurnTime : REAL);
    ASK METHOD ResetLighterStats;

    TELL METHOD ApproachAndMoor(IN index : INTEGER;
                                IN Dest : DestinationType;
                                IN Obj : ANYOBJ);
    TELL METHOD OnLoad(IN RoRo : RoRoObj);
    TELL METHOD CastAndClear(IN Berth : DestinationType;
                             IN Obj : ANYOBJ);
    TELL METHOD TransitTo(IN Dest : DestinationType;
                          IN Obj : ANYOBJ);
    TELL METHOD OffLoad(IN Obj : ANYOBJ);
    TELL METHOD Refuel(IN Obj : ANYOBJ);

END OBJECT;

VAR

Lighter : LighterObj;

END MODULE.
```

```
IMPLEMENTATION MODULE Lighter;

{-------------------------------------------------------------
Module Name: Lighter                    Last Modified: 18 Jun 93
Author:    J. S. Noel
           Lt.     USN

DESCRIPTION:  Implements a Lighter (Boat) object.
-------------------------------------------------------------}
FROM MathMod     IMPORT EXP;
FROM Ship        IMPORT RoRoObj, ALL ShipTypeType;
FROM Beach       IMPORT BeachObj;
FROM Refuel      IMPORT RefuelAreaObj, RefuelArea;
FROM SLCP        IMPORT SLCP;
FROM BLCP        IMPORT BLCP;
FROM FuelCP      IMPORT FuelCP;
FROM SimMod      IMPORT SimTime;
FROM Global      IMPORT ALL DestinationType, ALL SpotType;
FROM Stats       IMPORT Stats;
FROM WriteLine   IMPORT WriteLine;
FROM Convert     IMPORT LighterNameTypeToStr, BeachTypeToStr;
FROM RepMngr     IMPORT RepManager;



OBJECT LighterObj;

{-------------------------------------------------------------}
ASK METHOD ObjInit;
{-------------------------------------------------------------}

BEGIN

TimeInShipQueue := 0.0;
TimeInBeachQueue := 0.0;

InShipQTime    := 0.0;
OutShipQTime   := 0.0;
InBeachQTime   := 0.0;
OutBeachQTime  := 0.0;
```

```
TimeToTotalOffload := 0.0;
LoadStatus := FALSE;
ShipSpotIndex := 0;
BeachSpotIndex := 0;
RefuelSpotIndex := 0;
TransitDistance := 0.0;


END METHOD;

{-------------------------------------------------------------}
ASK METHOD GetLighterSetup(IN ID : STRING;
                           IN Name : LighterNameType;
                           IN Sp : SpotType;
                           IN Num1 : REAL;
                           IN Num2 : REAL;
                           IN Num3 : INTEGER;
                           IN Num4 : REAL;
                           IN Num5 : REAL;
                           IN Num6 : REAL);

{-------------------------------------------------------------}

BEGIN

LighterID       :=ID;
LighterTypeName :=Name;
LighterSpot     :=Sp;

{
WriteLine("//GetLighterSetup//  LighterTypeName = " +
         LighterNameTypeToStr(LighterTypeName) +
         " LighterID = " + LighterID);
}

MaxSpeed        :=Num1;
FullLoadSpeed   :=Num2;
MyLoadSize      :=Num3;
```

```
FuelCapacity     :=Num4;
FuelBurnRate     :=Num5;
MinFuelPercent   :=Num6;

CurrentFuel := FuelCapacity;
MinFuel := FuelCapacity * MinFuelPercent;
WriteLine("  MinFuel = " + REALTOSTR(MinFuel));


END METHOD;

{-------------------------------------------------------------}
ASK METHOD LogQueueTime(IN InOutQTime : REAL;
                        IN EnterQ : BOOLEAN;
                        IN ShipQ : BOOLEAN);
{-------------------------------------------------------------}

BEGIN

{
WriteLine("LogQueueTime fired on " +
           LighterNameTypeToStr(LighterTypeName) +
          "LighterID = " + LighterID);
}

   IF (EnterQ) AND (ShipQ)
      InShipQTime   := 0.0;
      InShipQTime := InOutQTime;
   ELSIF (NOT EnterQ) AND (ShipQ)
      OutShipQTime  := 0.0;
      OutShipQTime := InOutQTime;
      ASK SELF TO BurnFuel(OutShipQTime - InShipQTime);
      TimeInShipQueue := TimeInShipQueue + (OutShipQTime -
                                            InShipQTime);
   ELSIF (EnterQ) AND (NOT ShipQ)
      InBeachQTime  := 0.0;
      InBeachQTime := InOutQTime;
   ELSE
      OutBeachQTime := 0.0;
      OutBeachQTime := InOutQTime;
```

```
      ASK SELF TO BurnFuel(OutBeachQTime - InBeachQTime);
      TimeInBeachQueue := TimeInBeachQueue + (OutBeachQTime
                          - InBeachQTime);
   END IF;

END METHOD;

{-------------------------------------------------------------}
ASK METHOD SetLoadStatus(IN Status : BOOLEAN;
                         IN Vessel : RoRoObj);
{-------------------------------------------------------------}

BEGIN

{
WriteLine("SetLoadStatus fired on " +
LighterNameTypeToStr(LighterTypeName) +
          "LighterID = " + LighterID);
}

LoadStatus := Status;
ShipID := Vessel;

END METHOD;


{-------------------------------------------------------------}
ASK METHOD BurnFuel(IN BurnTime : REAL);
{-------------------------------------------------------------}

BEGIN

CurrentFuel := CurrentFuel - (FuelBurnRate * BurnTime/60.0);

{
WriteLine("Burning fuel.  CurrentFuel = " +
REALTOSTR(CurrentFuel));
WriteLine("  ");
}
```

```
END METHOD;

{-------------------------------------------------------------}
ASK METHOD ResetLighterStats;
{-------------------------------------------------------------}

BEGIN

{
WriteLine("ResetLighterStats " + LighterID);
}
InShipQTime    := 0.0;
OutShipQTime   := 0.0;
InBeachQTime   := 0.0;
OutBeachQTime  := 0.0;

TimeInShipQueue := 0.0;
TimeInBeachQueue := 0.0;
TimeToTotalOffload := 0.0;
LoadStatus := FALSE;
ShipSpotIndex := 0;
BeachSpotIndex := 0;
RefuelSpotIndex := 0;
TransitDistance := 0.0;
CurrentFuel := FuelCapacity;

END METHOD;

{-------------------------------------------------------------}
TELL METHOD ApproachAndMoor(IN index : INTEGER;
                            IN Dest : DestinationType;
                            IN Obj  : ANYOBJ);
{-------------------------------------------------------------}

VAR
ApproachAndMoorTime : REAL;
OperationalDelay    : REAL;
RoRo                : RoRoObj;
Beach               : BeachObj;
RefuelArea          : RefuelAreaObj;
```

```
MeanCWF                 : REAL;
MeanLCU                 : REAL;


BEGIN

IF ( Dest = Ship )
   ShipSpotIndex := index;

   RoRo := Obj;

   IF ( ASK RoRo ShipType = SSR )
      MeanCWF := 10.5;
      MeanLCU :=14.25;

   ELSE { ShipType = NSSR }
      MeanCWF :=8.0;
      MeanLCU :=14.25;

   END IF;


   IF LighterSpot = CWF
      ApproachAndMoorTime := ASK RandTime1 Normal(MeanCWF,
3.22);
      OperationalDelay := ASK RandTime2 Normal(2.0, 0.85);
   ELSE { LighterSpot = LCU }
      ApproachAndMoorTime := ASK RandTime1 Normal(MeanLCU,
2.22);
      OperationalDelay := ASK RandTime2 Normal(2.0, 0.85);

   END IF;

ELSIF ( Dest = Bch )
   BeachSpotIndex := index;

   IF LighterSpot = CWF
      ApproachAndMoorTime :=ASK RandTime1 Normal(17.0,3.43);
      OperationalDelay :=EXP(ASK RandTime2
                             Normal(1.0,0.85));
```

```
     ELSE { LighterSpot = LCU }
       ApproachAndMoorTime :=ASK RandTime1
                                  Normal(11.0,4.298);
       OperationalDelay :=ASK RandTime2 UniformReal(1.0,3.0);

    END IF;

ELSE

{Dest = RefuelArea}

    RefuelSpotIndex := index;

    IF LighterSpot = CWF
       ApproachAndMoorTime :=ASK RandTime1 Normal(17.0,3.43);
       OperationalDelay := ASK RandTime2 Normal(1.0, 0.85);
    ELSE  { LighterSpot = LCU }
       ApproachAndMoorTime :=ASK RandTime1
                                  Normal(11.0,4.298);
       OperationalDelay :=ASK RandTime2 UniformReal(1.0,3.0);

    END IF;

END IF;


WAIT DURATION ApproachAndMoorTime + OperationalDelay
END WAIT;


ASK SELF TO BurnFuel(ApproachAndMoorTime +
                     OperationalDelay);


IF ( Dest = Ship )
{
    WriteLine("ApproachAndMoor Ship " + " ShipSpot = " +
              INTTOSTR(ShipSpotIndex) + " LighterID = " +
              LighterID);
    WriteLine(" ApproachAndMoorTime = " +
```

```
            REALTOSTR(ApproachAndMoorTime +
            OperationalDelay));
}

   ASK RoRo TO MakeLoad(SELF, ShipSpotIndex);
ELSIF ( Dest = Bch )
{
   WriteLine("ApproachAndMoor Beach " + " BeachSpot = " +
            INTTOSTR(BeachSpotIndex) + " LighterID = " +
            LighterID);
   WriteLine(" ApproachAndMoorTime = " +
            REALTOSTR(ApproachAndMoorTime +
            OperationalDelay));
 }

   Beach := Obj;
   ASK Beach TO OccupyBeachSpot(BeachSpotIndex);
   TELL SELF TO OffLoad(Beach);
ELSE
{Dest = RefuelArea}
{
  WriteLine("ApproachAndMoor RefuelArea " + " RefuelAreaSpot
            = " + INTTOSTR(RefuelSpotIndex) +
            " LighterID = " + LighterID);
   WriteLine(" ApproachAndMoorTime = " +
            REALTOSTR(ApproachAndMoorTime +
            OperationalDelay));
}

  RefuelArea := Obj;
  TELL SELF TO Refuel(RefuelArea);
END IF;


END METHOD;


{-------------------------------------------------------------}
TELL METHOD OnLoad(IN RoRo : RoRoObj);
{-------------------------------------------------------------}
```

```
VAR
OnLoadTime          : REAL;
OperationalDelay2   : REAL;
MeanCWF             : REAL;
MeanLCU             : REAL;


BEGIN


IF ( ASK RoRo ShipType = SSR )
   MeanCWF := 16.0;
   MeanLCU := 15.85;

ELSE { ShipType = NSSR }
   MeanCWF := 25.0;
   MeanLCU := 18.0;

END IF;

IF LighterSpot = CWF
   OnLoadTime := ASK RandTime1 Normal(MeanCWF, 3.87);
   OperationalDelay2 :=EXP(ASK RandTime2
                                Normal(1.24,1.186));
   OnLoadTime := OnLoadTime * FLOAT(MyLoadSize);
ELSIF LighterSpot = LCU
   OnLoadTime := ASK RandTime1 Normal(MeanLCU, 3.87);
   OperationalDelay2 := ASK RandTime2 Normal(1.0, 0.42);
   OnLoadTime := OnLoadTime * FLOAT(MyLoadSize);
ELSE
   OnLoadTime := ASK RandTime2 Normal(10.25, 5.75);
   OnLoadTime := OnLoadTime * FLOAT(MyLoadSize);

END IF;


WAIT DURATION OnLoadTime + OperationalDelay2
END WAIT;
```

```
{
WriteLine("Onload fired " +
          LighterNameTypeToStr(LighterTypeName) +
          " LighterID = " + LighterID + " Ship is " + ASK
          RoRo ShipName);
WriteLine("OnLoadTime = " + REALTOSTR(OnLoadTime +
          OperationalDelay2));
WriteLine("  ");
}


ASK SELF TO BurnFuel(OnLoadTime + OperationalDelay2);

TELL SELF TO CastAndClear(Ship, RoRo);


END METHOD;

{-------------------------------------------------------------}
TELL METHOD CastAndClear(IN Berth : DestinationType;
                         IN Obj  : ANYOBJ);
{-------------------------------------------------------------}

VAR
CastAndClearTime     : REAL;
RoRo                 : RoRoObj;
Beach                : BeachObj;
RefuelArea           : RefuelAreaObj;
MeanCWF              : REAL;
MeanLCU              : REAL;
Dest                 : DestinationType;



BEGIN


IF ( Berth = Ship )
   RoRo := Obj;
```

```
   IF ( ASK RoRo ShipType = SSR )
      MeanCWF := 5.0;
      MeanLCU :=2.0;

   ELSE { ShipType = NSSR }
      MeanCWF :=4.0;
      MeanLCU :=4.0;

   END IF;

   IF LighterSpot = CWF
      CastAndClearTime :=ASK RandTime1 Normal(MeanCWF,1.33);
   ELSE { LighterSpot = LCU }
      CastAndClearTime :=ASK RandTime1
                           UniformReal(MeanLCU, 2.5);

   END IF;

ELSIF ( Berth = Bch )
   Beach := Obj;

   IF LighterSpot = CWF
       CastAndClearTime := ASK RandTime1 Normal(9.9, 1.76);
   ELSE { LighterSpot = LCU }
      CastAndClearTime :=ASK RandTime1
                          UniformReal(1.75,3.0);

   END IF;

ELSE

{ Berth = RefuelArea }

   IF LighterSpot = CWF
      CastAndClearTime := ASK RandTime1 Normal(9.9, 1.76);
   ELSE { LighterSpot = LCU }
      CastAndClearTime :=ASK RandTime1
                           UniformReal(1.75, 3.0);

   END IF;
```

```
END IF;

WAIT DURATION CastAndClearTime
END WAIT;

ASK SELF TO BurnFuel(CastAndClearTime);

IF ( Berth = Ship )

   {If this Lighter has the last load, LoadStatus = T, then
    the Lighter simply transits.  If this is not the last
    load, then the RoRo sets a Spot free   thus starting the
    whole lighter cycle for the next Lighter of the
    appropriate type in the AwaitingShipQueue.}

{
   WriteLine(LighterNameTypeToStr(LighterTypeName) + "
             LighterID = " +
             LighterID + " CastAndClear Ship " + ASK RoRo
             ShipName);
   WriteLine("CastAndClearTime = " +
             REALTOSTR(CastAndClearTime));
   WriteLine("  ");
}


   Dest := Bch;

   IF LoadStatus
      TELL SELF TO TransitTo(Dest, RoRo);
   ELSE
      ASK RoRo TO SetSpotFree(ShipSpotIndex);
      TELL SELF TO TransitTo(Dest, RoRo);
   END IF;

ELSIF ( Berth = Bch )

   {After Offload and CastAndClear are complete, check fuel
    status (only one check in cycle).  If less than or equal
```

```
    to the minimum allowable, the lighter must transit to
    the refueling area.  If LoadStatus = T, then lighter has
    just carried the last load to the beach and it is
    returning to the ShipQ.}

{
   WriteLine(LighterNameTypeToStr(LighterTypeName) + "
             LighterID = " +
             LighterID + " CastAndClear Beach " +
             BeachTypeToStr(ASK Beach BeachMake));
   WriteLine("CastAndClearTime = " +
             REALTOSTR(CastAndClearTime));
   WriteLine("  ");
}



   IF LoadStatus
      Dest := Ship;
      TELL SELF TO TransitTo(Dest, Beach);
      RETURN;

   ELSE
      IF (CurrentFuel <= MinFuel)
         Dest := Fuel;
         ASK Beach TO SetSpotFree(BeachSpotIndex);
         TELL SELF TO TransitTo(Dest, Beach);
      ELSE
         Dest := Ship;
         ASK Beach TO SetSpotFree(BeachSpotIndex);

{
         WriteLine("//Fuel Status// MinFuel = " +
                   REALTOSTR(MinFuel) +
                   " CurrentFuel = " +
                   REALTOSTR(CurrentFuel));
}

         TELL SELF TO TransitTo(Dest, Beach);
      END IF;
```

```
   END IF;

ELSE

   { Berth = RefuelArea }

   Dest := ShipFromRefuel;
   RefuelArea := Obj;

{
   WriteLine(LighterNameTypeToStr(LighterTypeName) + "
             LighterID = " + LighterID + " CastAndClear
             RefuelArea " + ASK RefuelArea AreaName);
   WriteLine("CastAndClearTime = " +
             REALTOSTR(CastAndClearTime));

   WriteLine("  ");
}

   ASK RefuelArea TO SetSpotFree(RefuelSpotIndex);
   TELL SELF TO TransitTo(Dest, RefuelArea);

END IF;


END METHOD;


{-------------------------------------------------------------}
TELL METHOD TransitTo(IN Dest : DestinationType;
                      IN Obj : ANYOBJ);
{-------------------------------------------------------------}

VAR

TransitTime : REAL;
RoRo        : RoRoObj;
Beach       : BeachObj;
RefuelArea  : RefuelAreaObj;
```

```
BEGIN

IF ( Dest = Bch )

  RoRo := Obj;

  TransitDistance := ASK RoRo DistanceFromBLCP;

  TransitTime := TransitDistance/(FullLoadSpeed *0.65)*60.0;

{
   WriteLine(LighterNameTypeToStr(LighterTypeName) + "
             LighterID = " +
             LighterID + " TransitTo fired, now leaving " +
             ASK RoRo ShipName + " for BLCP. ");
   WriteLine("TransitTime = " + REALTOSTR(TransitTime));
   WriteLine("    ");
}

ELSIF ( Dest = Ship )

   Beach := Obj;

   TransitDistance := ASK Beach DistanceToSLCP;

   TransitTime := TransitDistance/(MaxSpeed * 0.65) * 60.0;

{
   WriteLine(LighterNameTypeToStr(LighterTypeName) + "
             LighterID = " +
             LighterID + " TransitTo Fired, now leaving " +
             BeachTypeToStr(ASK Beach BeachMake) + " For
             SLCP.");
   WriteLine("TransitTime = " + REALTOSTR(TransitTime));
   WriteLine("    ");
}

ELSIF ( Dest = Fuel )
```

```
   Beach := Obj;
   Dest := RefuelFromBeach;

   TransitDistance := ASK Beach DistFromBeachToRefuel;
   TransitTime := TransitDistance/(MaxSpeed * 0.65) * 60.0;

{
   WriteLine(LighterNameTypeToStr(LighterTypeName) + "
             LighterID = " +
             LighterID + " TransitTo fired, now leaving
             beach for RefuelArea ");
   WriteLine("TransitTime = " + REALTOSTR(TransitTime));
   WriteLine("    ");
}

ELSE

{ Dest = ShipFromRefuel }

   RefuelArea := Obj;

   TransitDistance := ASK RefuelArea DistFromRefuelToShip;

   TransitTime := TransitDistance/(MaxSpeed * 0.65) * 60.0;

{
   WriteLine(LighterNameTypeToStr(LighterTypeName) + "
             LighterID = " +
             LighterID + " TransitTo fired, now leaving " +
             ASK RefuelArea AreaName + " for SLCP ");
   WriteLine("TransitTime = " + REALTOSTR(TransitTime));
   WriteLine("    ");
}

END IF;


WAIT DURATION TransitTime
END WAIT;
```

```
ASK SELF TO BurnFuel(TransitTime);


IF ( Dest = Bch )
   ASK BLCP TO GetSpot(SELF);
ELSIF ( Dest = Ship )
   ASK SLCP TO GetSpot(SELF);
ELSIF ( Dest = RefuelFromBeach )
   ASK FuelCP TO GetFuelSpot(SELF);
ELSE
{ Dest = ShipFromRefuel }
   ASK SLCP TO GetSpot(SELF);
END IF;


END METHOD;


{-------------------------------------------------------------}
TELL METHOD OffLoad(IN Obj : ANYOBJ);
{-------------------------------------------------------------}

VAR
OffLoadTime          : REAL;
OperationalDelay4    : REAL;
Berth                : DestinationType;
Beach                : BeachObj;

BEGIN


IF LighterSpot = CWF
   OffLoadTime := ASK RandTime1 Normal(10.0, 3.43);
   OperationalDelay4 := ASK RandTime2 UniformReal(0.0, 1.5);
   OffLoadTime := OffLoadTime * FLOAT(MyLoadSize);
ELSE { LighterSpot = LCU }
   OffLoadTime := ASK RandTime1 Normal(3.0, 0.859);
   OperationalDelay4 := ASK RandTime2 Normal(1.0, 0.333);
   OffLoadTime := OffLoadTime * FLOAT(MyLoadSize);
```

```
END IF;

WAIT DURATION OffLoadTime + OperationalDelay4
END WAIT;

Beach := Obj;

{
WriteLine(LighterNameTypeToStr(LighterTypeName) + "
          LighterID = " +
          LighterID + " Offload fired at " +
          BeachTypeToStr(ASK Beach BeachMake));
WriteLine("OffLoadTime = " + REALTOSTR(OffLoadTime +
           OperationalDelay4));
WriteLine("    ");
}


ASK SELF TO BurnFuel(OffLoadTime + OperationalDelay4);

Berth := Bch;

TELL SELF TO CastAndClear(Berth, Beach);

END METHOD;


{-------------------------------------------------------------}
TELL METHOD Refuel(IN Obj : ANYOBJ);
{-------------------------------------------------------------}

VAR
RefuelTime : REAL;
Rate : REAL;
RefuelArea : RefuelAreaObj;

BEGIN

{
```

```
WriteLine(LighterNameTypeToStr(LighterTypeName) + "
LighterID = " +
          LighterID + " Refueling ");
}

RefuelArea := Obj;

Rate := ASK RefuelArea PumpRate;
RefuelTime := ((FuelCapacity - CurrentFuel) / Rate) * 60.0;
CurrentFuel := FuelCapacity;

{
WriteLine("RefuelTime = " + REALTOSTR(RefuelTime));
WriteLine("    ");
}

WAIT DURATION RefuelTime
END WAIT;

TELL SELF TO CastAndClear(Fuel, Obj);

END METHOD;


END OBJECT;

END MODULE.



DEFINITION MODULE BLCP;

{-------------------------------------------------------------
Module Name: BLCP                    Last Modified: 18 Jun 93
Author:  J. S. Noel
         Lt.    USN

DESCRIPTION:  Defines the Beach Lighterage Control Point
Object (BLCPObj).  After a Lighter CastsandClears the beach
```

```
it asks the BeachObj to SetSpotFree.  This method fires the
GetLighter method in BLCPObj.  BLCPObj then pops the first
appropriate lighter off of theAwaitingBeachQueue and directs
the lighter to  ApproachAndMoor to the Beach, where the
offload of vehicles can begin.
-------------------------------------------------------------}
FROM GrpMod  IMPORT QueueObj;
FROM Lighter IMPORT LighterObj;
FROM Beach   IMPORT BeachObj;

TYPE

AwaitingBeachQueueObj = OBJECT(QueueObj[ANYOBJ: LighterObj])
END OBJECT;

BLCPObj = OBJECT

   ASK METHOD ObjInit;
   ASK METHOD GetSpot(IN Lighter : LighterObj);
   ASK METHOD GetLighter(IN index : INTEGER;
                         IN Beach : BeachObj);

END OBJECT;

VAR

WaitForBeachQueue : AwaitingBeachQueueObj;
BLCP : BLCPObj;

END MODULE.



IMPLEMENTATION MODULE BLCP;

{-------------------------------------------------------------
Module Name: BLCP                      Last Modified: 17 Jul 93
Author:   J. S. Noel
          Lt.    USN
```

```
DESCRIPTION:  Implements a Ship Lighter Control Point (SLCP)
object.
-------------------------------------------------------------}
FROM Global    IMPORT SpotType,
                      ALL DestinationType;
FROM Lighter   IMPORT LighterObj;
FROM Beach     IMPORT BeachObj;
FROM SimMod    IMPORT SimTime;
FROM Builder   IMPORT BeachBuilder;
FROM WriteLine IMPORT WriteLine;


OBJECT BLCPObj;

{-------------------------------------------------------------}
ASK METHOD ObjInit;
{-------------------------------------------------------------}

BEGIN

NEW(WaitForBeachQueue);

END METHOD;

{-------------------------------------------------------------}
ASK METHOD GetSpot(IN Lighter : LighterObj);
{-------------------------------------------------------------}

VAR
i          : INTEGER;
SpotAvail  : BOOLEAN;
LogIn      : BOOLEAN;
ThisQ      : BOOLEAN;
TheBeach   : BeachObj;
Dest       : DestinationType;

BEGIN
{
WriteLine("GetSpot Fired in BLCP ");
}
```

```
SpotAvail := FALSE;
Dest := Bch;

TheBeach := ASK BeachBuilder First();
WHILE TheBeach <> NILOBJ
   ASK TheBeach TO CheckSpots(SpotAvail, i);

   IF SpotAvail
      TELL Lighter TO ApproachAndMoor(i, Dest, TheBeach);
      EXIT;

   END IF;

TheBeach := ASK BeachBuilder Next(TheBeach);
END WHILE;

IF ( NOT SpotAvail )
   ASK WaitForBeachQueue TO Add(Lighter);
   LogIn := TRUE; { adding to Q }
   ThisQ := FALSE; { Q type = Beach }
   ASK Lighter TO LogQueueTime(SimTime(), LogIn, ThisQ);
END IF;

END METHOD;

{-------------------------------------------------------------}
ASK METHOD GetLighter(IN index : INTEGER;
                      IN Beach : BeachObj);
{-------------------------------------------------------------}

VAR
Lighter : LighterObj;
LogIn : BOOLEAN;
ThisQ : BOOLEAN;
Dest      : DestinationType;


BEGIN
{
```

```
WriteLine("GetLighter Fired in BLCP ");
}

Dest := Bch;

Lighter := ASK WaitForBeachQueue First();
IF Lighter <> NILOBJ

   ASK WaitForBeachQueue TO RemoveThis(Lighter);
   LogIn := FALSE; { not adding to Q }
   ThisQ := FALSE; { Q type = beach }
   ASK Lighter TO LogQueueTime(SimTime(), LogIn, ThisQ);
   TELL Lighter TO ApproachAndMoor(index, Dest, Beach);
   ASK Beach TO OccupyBeachSpot(index);


END IF;


END METHOD;

END OBJECT;

END MODULE.



DEFINITION MODULE Beach;

{-------------------------------------------------------------
Module Name: Beach                    Last Modified: 20 Jul 93
Author:    J. S. Noel
           Lt.     USN

DESCRIPTION:  Defines a Beach object.
-------------------------------------------------------------}
FROM ShpList IMPORT SpotArrayType;
FROM Global  IMPORT SpotIdleTimeArrayType;
```

```
TYPE

BeachType = (BareBeach, FloatingCWPier, ELCAS);

BeachObj = OBJECT

   BeachMake              : BeachType;
   BeachID                : STRING;
   NumSpots               : INTEGER;
   BeachSpot              : SpotArrayType;
   DistanceToSLCP         : REAL;
   DistFromBeachToRefuel  : REAL;
   BeachSpotIdleTime      : SpotIdleTimeArrayType;

   ASK METHOD ObjInit;
   ASK METHOD GetBeachSetup(IN ID : STRING;
                            IN Name : BeachType;
                            IN Num1 : INTEGER;
                            IN Array : SpotArrayType;
                            IN Num2 : REAL;
                            IN Num3 : REAL);

   ASK METHOD SetSpotFree(IN index : INTEGER);
   ASK METHOD CheckSpots(OUT SpotAvail : BOOLEAN;
                         OUT index : INTEGER);
   ASK METHOD LogIdleBeachSpotTime(IN index : INTEGER;
                                   IN InOutSpotTime : REAL;
                                   IN SpotIdle : BOOLEAN);
   ASK METHOD OccupyBeachSpot(IN index : INTEGER);
   ASK METHOD ResetBeachStats;


END OBJECT;

VAR
Beach : BeachObj;

END MODULE.
```

```
IMPLEMENTATION MODULE Beach;

{-------------------------------------------------------------
Module Name: Beach                      Last Modified: 18 Jun 93
Author:    J. S. Noel
           Lt.     USN

DESCRIPTION:  Implements a Beach object.
-------------------------------------------------------------}
FROM ShpList    IMPORT SpotArrayType;
FROM Global     IMPORT ALL SpotType, SpotRecType,
                       ALL SpotIdleTimeRecType;
FROM SimMod     IMPORT SimTime;
FROM BLCP       IMPORT BLCP;
FROM WriteLine IMPORT WriteLine;
FROM Convert    IMPORT BeachTypeToStr;

TYPE

OBJECT BeachObj;

{-------------------------------------------------------------}
ASK METHOD ObjInit;
{-------------------------------------------------------------}

BEGIN

END METHOD;


{-------------------------------------------------------------}
ASK METHOD GetBeachSetup(IN ID : STRING;
                         IN Name : BeachType;
                         IN Num1 : INTEGER;
                         IN Array : SpotArrayType;
                         IN Num2 : REAL;
                         IN Num3 : REAL);

{-------------------------------------------------------------}
```

```
VAR
i : INTEGER;
Rec : SpotIdleTimeRecType;

BEGIN

BeachID         := ID;

BeachMake       := Name;
{
WriteLine("//GetBeachSetup//  BeachMake = " +
          BeachTypeToStr(BeachMake));
}

NumSpots        := Num1;

NEW(BeachSpot, 1..NumSpots);
BeachSpot       := Array;

DistanceToSLCP := Num2;
DistFromBeachToRefuel := Num3;

NEW(BeachSpotIdleTime, 1..NumSpots);

FOR i := 1 TO NumSpots
   BeachSpot[i].TotalIdleTime := 0.0;

   NEW(Rec);
   Rec.StartTime := 0.0;
   Rec.EndTime := 0.0;
   BeachSpotIdleTime[i] := Rec;
END FOR;


END METHOD;
```

```
{-------------------------------------------------------}
ASK METHOD LogIdleBeachSpotTime(IN index : INTEGER;
                                IN InOutSpotTime : REAL;
                                IN SpotIdle : BOOLEAN);
{-------------------------------------------------------}

BEGIN
{
WriteLine("LogIdleBeachSpotTime Fired ");
}

IF SpotIdle
   BeachSpotIdleTime[index].StartTime := InOutSpotTime;
ELSE
   BeachSpotIdleTime[index].EndTime := InOutSpotTime;
   BeachSpot[index].TotalIdleTime :=
BeachSpot[index].TotalIdleTime +

(BeachSpotIdleTime[index].EndTime -

BeachSpotIdleTime[index].StartTime);

END IF;

END METHOD;


{-------------------------------------------------------}
ASK METHOD SetSpotFree(IN index : INTEGER);
{-------------------------------------------------------}

VAR
Idle : BOOLEAN;

BEGIN

{
WriteLine("SetSpotFree fired in beach " +
BeachTypeToStr(BeachMake) + " Spot " +
          INTTOSTR(index));
```

```
}

BeachSpot[index].SpotFree := TRUE;
Idle := BeachSpot[index].SpotFree;
ASK SELF TO LogIdleBeachSpotTime(index, SimTime(), Idle);
ASK BLCP TO GetLighter(index, SELF);

END METHOD;

{-------------------------------------------------------------}
ASK METHOD CheckSpots(OUT SpotAvail : BOOLEAN;
                      OUT index : INTEGER);
{-------------------------------------------------------------}

BEGIN
{
WriteLine("CheckSpots fired in beach " +
BeachTypeToStr(BeachMake));
}

SpotAvail := FALSE;

FOR index := 1 TO HIGH(BeachSpot)
   IF (BeachSpot[index].SpotFree)

      SpotAvail := TRUE;
      BeachSpot[index].SpotFree := FALSE;
      EXIT;
   END IF;
END FOR;

END METHOD;

{-------------------------------------------------------------}
ASK METHOD OccupyBeachSpot(IN index : INTEGER);
{-------------------------------------------------------------}

VAR
Idle : BOOLEAN;
```

```
BEGIN
{
WriteLine("Occupy spot fired in beach " +
BeachTypeToStr(BeachMake));
}

BeachSpot[index].SpotFree := FALSE;
Idle := BeachSpot[index].SpotFree;

ASK SELF TO LogIdleBeachSpotTime(index, SimTime(), Idle);

END METHOD;

{-------------------------------------------------------------}
ASK METHOD ResetBeachStats;
{-------------------------------------------------------------}

VAR
i : INTEGER;

BEGIN
{
WriteLine("ResetBeachStats " + BeachTypeToStr(BeachMake));
}

FOR i := 1 TO NumSpots
   BeachSpot[i].SpotFree := TRUE;
   BeachSpot[i].TotalIdleTime := 0.0;
   BeachSpotIdleTime[i].StartTime := 0.0;
   BeachSpotIdleTime[i].EndTime := 0.0;
END FOR;

END METHOD;


END OBJECT;

END MODULE.
```

```
DEFINITION MODULE FuelCP;

{-------------------------------------------------------------
Module Name: FuelCP                    Last Modified: 20 Jul 93
Author:   J. S. Noel
          Lt.    USN

DESCRIPTION:  Defines the Fuel Control Point Object
(FuelCPObj) and the WaitingForFuelQueueObj.  The
WaitingForFuelQueue is a FIFO group of lighters waiting for
an empty spot for refueling.  When a spot opens up the
FuelCPObj pops the first lighter off of the queue and TELLs
it ApproachAndMoorRefuelArea.
-------------------------------------------------------------}
FROM GrpMod      IMPORT QueueObj;
FROM Global      IMPORT SpotType;
FROM Lighter     IMPORT LighterObj;
FROM Refuel      IMPORT RefuelAreaObj;

TYPE

WaitingForFuelQueueObj = OBJECT(QueueObj[ANYOBJ:
LighterObj])
END OBJECT;

FuelCPObj = OBJECT

   ASK METHOD ObjInit;
   ASK METHOD GetFuelSpot(IN Lighter : LighterObj);
   ASK METHOD GetGasLowLighter(IN index : INTEGER;
                               IN RefuelArea : RefuelAreaObj);

END OBJECT;

VAR
WaitForFuelQueue : WaitingForFuelQueueObj;
FuelCP : FuelCPObj;
```

```
END MODULE.


IMPLEMENTATION MODULE FuelCP;

{-------------------------------------------------------------
Module Name: FuelCP                    Last Modified: 20 Jul 93
Author:   J. S. Noel
          Lt.    USN

DESCRIPTION:  Implements the Fuel Control Point Object
(FuelCPObj) and the WaitingForFuelQueueObj.
-------------------------------------------------------------}
FROM Lighter    IMPORT LighterObj;
FROM SimMod     IMPORT SimTime;
FROM Refuel     IMPORT RefuelAreaObj;
FROM Builder    IMPORT FuelAreaBuilder;
FROM Global     IMPORT ALL DestinationType;
FROM WriteLine  IMPORT WriteLine;

TYPE

OBJECT FuelCPObj;

{-------------------------------------------------------------}
ASK METHOD ObjInit;
{-------------------------------------------------------------}

BEGIN

NEW(WaitForFuelQueue);

END METHOD;

{-------------------------------------------------------------}
ASK METHOD GetFuelSpot(IN Lighter : LighterObj);
{-------------------------------------------------------------}

VAR
```

```
SpotAvail   : BOOLEAN;
index       : INTEGER;
RefuelArea  : RefuelAreaObj;
Dest        : DestinationType;


BEGIN
{
WriteLine("GetFuelSpot Fired " + " LighterID = " + ASK
Lighter LighterID);
}

SpotAvail := FALSE;
Dest := Fuel;

RefuelArea := ASK FuelAreaBuilder First();
WHILE RefuelArea <> NILOBJ
   ASK RefuelArea TO CheckFuelSpots(SpotAvail, index);

   IF SpotAvail
      TELL Lighter TO ApproachAndMoor(index, Dest,
RefuelArea);
      ASK RefuelArea TO OccupySpot(index);
      RETURN;
   END IF;

RefuelArea := ASK FuelAreaBuilder Next(RefuelArea);
END WHILE;

IF (NOT SpotAvail )
   ASK WaitForFuelQueue TO Add(Lighter);

END IF;

END METHOD;

{-------------------------------------------------------}
ASK METHOD GetGasLowLighter(IN index : INTEGER;
                            IN RefuelArea : RefuelAreaObj);
{-------------------------------------------------------}
```

```
VAR
Lighter : LighterObj;
Dest    : DestinationType;

BEGIN
{
WriteLine("GetGasLowLighter Fired ");
}

Dest := Fuel;

Lighter := ASK WaitForFuelQueue First();

IF Lighter <> NILOBJ

   ASK WaitForFuelQueue TO RemoveThis(Lighter);

   TELL Lighter TO ApproachAndMoor(index, Dest, RefuelArea);

END IF;

END METHOD;

END OBJECT;

END MODULE.


DEFINITION MODULE Refuel;

{-------------------------------------------------------------
Module Name: RefuelArea            Last Modified: 20 Jul 93
Author:   J. S. Noel
          Lt.    USN

DESCRIPTION:  Defines The RefuelArea object.
-------------------------------------------------------------}
```

```
FROM Global  IMPORT RefuelSpotRecType;
FROM Lighter IMPORT LighterObj;

TYPE

RefuelSpotArrayType = ARRAY INTEGER OF RefuelSpotRecType;

RefuelAreaObj = OBJECT

   AreaName                : STRING;
   NumRefuelSpots          : INTEGER;
   DistFromRefuelToShip    : REAL;
   RefuelSpot              : RefuelSpotArrayType;
   PumpRate                : REAL;

   ASK METHOD ObjInit;
   ASK METHOD GetRefuelAreaSetup(IN Name : STRING;
                                 IN Num1 : INTEGER;
                                 IN Num2 : REAL;
                                 IN Array :
RefuelSpotArrayType;
                                 IN Num3 : REAL);

   ASK METHOD SetSpotFree(IN index : INTEGER);
   ASK METHOD CheckFuelSpots(OUT SpotAvail : BOOLEAN;
                             OUT index : INTEGER);
   ASK METHOD OccupyRefuelSpot(IN index : INTEGER);


END OBJECT;

VAR
RefuelArea : RefuelAreaObj;

END MODULE.



IMPLEMENTATION MODULE Refuel;
```

```
{-------------------------------------------------------------
Module Name: RefuelArea                Last Modified: 20 Jul 93
Author:    J. S. Noel
           Lt.     USN

DESCRIPTION:  Implements The RefuelArea object.
-------------------------------------------------------------}
FROM FuelCP    IMPORT FuelCP;
FROM Lighter   IMPORT LighterObj;
FROM WriteLine IMPORT WriteLine;


OBJECT RefuelAreaObj;




{-------------------------------------------------------------}
ASK METHOD ObjInit;
{-------------------------------------------------------------}

BEGIN


END METHOD;


{-------------------------------------------------------------}
ASK METHOD GetRefuelAreaSetup(IN Name : STRING;
                              IN Num1 : INTEGER;
                              IN Num2 : REAL;
                              IN Array :
RefuelSpotArrayType;
                              IN Num3 : REAL);
{-------------------------------------------------------------}

BEGIN
```

```
AreaName                :=Name;

{
WriteLine("//GetRefuelAreaSetup//  AreaName = " + AreaName);
}

NumRefuelSpots          :=Num1;
DistFromRefuelToShip    :=Num2;

NEW(RefuelSpot, 1..NumRefuelSpots);
RefuelSpot              :=Array;

PumpRate                :=Num3;



END METHOD;



{-------------------------------------------------------------}
ASK METHOD SetSpotFree(IN index : INTEGER);
{-------------------------------------------------------------}

BEGIN

{
WriteLine("SetSpotFree Fired in RefuelArea ");
}

RefuelSpot[index].RefuelSpotFree := TRUE;
ASK FuelCP TO GetGasLowLighter(index, SELF);

END METHOD;
```

```
{-------------------------------------------------------------}
ASK METHOD CheckFuelSpots(OUT SpotAvail : BOOLEAN;
                          OUT index : INTEGER);
{-------------------------------------------------------------}


BEGIN

{
WriteLine("CheckFuelSpots fired in RefuelArea ");
}

SpotAvail := FALSE;

FOR index := 1 TO HIGH(RefuelSpot)
   IF (RefuelSpot[index].RefuelSpotFree)
      SpotAvail := TRUE;
      RETURN;
   END IF;
END FOR;

END METHOD;

{-------------------------------------------------------------}
ASK METHOD OccupyRefuelSpot(IN index : INTEGER);
{-------------------------------------------------------------}
BEGIN

{
WriteLine("OccupySpot Fired in RefuelArea spot = " +
          INTTOSTR(index));
}

RefuelSpot[index].RefuelSpotFree := FALSE;

END METHOD;

END OBJECT;

END MODULE.
```

```
DEFINITION MODULE Convert;

{-------------------------------------------------------------
Module Name: Convert                    Last Modified: 20 Jul 93
Author:      J. S. Noel
              Lt.    USN

DESCRIPTION:  Defines Procedures for converting enumerated
types to/from STRING for input/output.
-------------------------------------------------------------}
FROM Global  IMPORT SpotType;
FROM Beach   IMPORT BeachType;
FROM Lighter IMPORT LighterNameType;
FROM Ship    IMPORT ShipTypeType;


PROCEDURE SpotTypeToStr(IN Spot : SpotType) : STRING;
PROCEDURE StrToSpotType(IN Str : STRING) : SpotType;

PROCEDURE BeachTypeToStr(IN BchName : BeachType) : STRING;
PROCEDURE StrToBeachType(IN Str : STRING) : BeachType;

PROCEDURE LighterNameTypeToStr(IN LighterName :
                               LighterNameType) : STRING;
PROCEDURE StrToLighterNameType(IN Str : STRING) :
                               LighterNameType;

PROCEDURE ShipTypeToStr(IN ShipType : ShipTypeType) :
                        STRING;
PROCEDURE StrToShipType(IN Str : STRING) : ShipTypeType;

PROCEDURE BooleanToStr(IN Boolean : BOOLEAN) : STRING;
PROCEDURE StrToBoolean(IN Str : STRING) : BOOLEAN;


END MODULE.


IMPLEMENTATION MODULE Convert;
```

```
{-------------------------------------------------------------
Module Name: Convert                    Last Modified: 20 Jul 93
Author:    J. S. Noel
           Lt.     USN

DESCRIPTION:  Implements Procedures for converting
enumerated types to/from STRING for input/output.
-------------------------------------------------------------}
FROM Global     IMPORT ALL SpotType;
FROM Beach      IMPORT ALL BeachType;
FROM Lighter    IMPORT ALL LighterNameType;
FROM Ship       IMPORT ALL ShipTypeType;
FROM WriteLine IMPORT WriteLine;




{-------------------------------------------------------------}
PROCEDURE SpotTypeToStr(IN Spot : SpotType) : STRING;
{-------------------------------------------------------------}

VAR
Str : STRING;

BEGIN

CASE Spot
   WHEN LCU : Str := "LCU";
   WHEN CWF : Str := "CWF";
   WHEN LoLo : Str := "LoLo";
   OTHERWISE
      Str := "Other...?";
END CASE;

RETURN(Str);

END PROCEDURE;
```

```
{-------------------------------------------------------------}
PROCEDURE StrToSpotType(IN Str : STRING) : SpotType;
{-------------------------------------------------------------}

VAR
Spot : SpotType;

BEGIN

WriteLine("Converting" + Str);

CASE Str
   WHEN "LCU" : Spot := LCU;
   WHEN "Lcu" : Spot := LCU;
   WHEN "lCU" : Spot := LCU;
   WHEN "CWF" : Spot := CWF;
   WHEN "Cwf" : Spot := CWF;
   WHEN "cwf" : Spot := CWF;
   WHEN "LoLo" : Spot := LoLo;
   WHEN "Lolo" : Spot := LoLo;
   WHEN "lolo" : Spot := LoLo;
   WHEN "LOLO" : Spot := LoLo;
END CASE;

RETURN(Spot);

END PROCEDURE;


{-------------------------------------------------------------}
PROCEDURE BeachTypeToStr(IN BchName : BeachType) : STRING;
{-------------------------------------------------------------}

VAR
Str : STRING;

BEGIN

CASE BchName
```

```
   WHEN BareBeach : Str := "BareBeach";
   WHEN FloatingCWPier : Str := "FloatingCWPier";
   WHEN ELCAS : Str := "ELCAS";
   OTHERWISE
      Str := "Other...?";
END CASE;

RETURN(Str);

END PROCEDURE;


{-------------------------------------------------------------}
PROCEDURE StrToBeachType(IN Str : STRING) : BeachType;
{-------------------------------------------------------------}

VAR
BeachName : BeachType;

BEGIN

WriteLine("Converting" + Str);

CASE Str
   WHEN "BareBeach" : BeachName := BareBeach;
   WHEN "FloatingCWPier" : BeachName := FloatingCWPier;
   WHEN "ELCAS" : BeachName := ELCAS;

END CASE;

RETURN(BeachName);

END PROCEDURE;


{-------------------------------------------------------------}
PROCEDURE LighterNameTypeToStr(IN LighterName :
                               LighterNameType) : STRING;
{-------------------------------------------------------------}
VAR
```

```
Str : STRING;

BEGIN

CASE LighterName
   WHEN LCU1466 : Str := "LCU1466";
   WHEN LCU1610 : Str := "LCU1610";
   WHEN LCU2000 : Str := "LCU2000";
   WHEN CWF11   : Str := "CWF11";
   WHEN CWF21   : Str := "CWF21";
   WHEN CWF31   : Str := "CWF31";
   WHEN LSV     : Str := "LSV";
   OTHERWISE
      Str := "Other...?";
END CASE;

RETURN(Str);

END PROCEDURE;

{-------------------------------------------------------------}
PROCEDURE StrToLighterNameType(IN Str : STRING) :
LighterNameType;
{-------------------------------------------------------------}

VAR
LighterName : LighterNameType;

BEGIN

WriteLine("Converting" + Str);

CASE Str
   WHEN "LCU1466" : LighterName := LCU1466;
   WHEN "LCU1610" : LighterName := LCU1610;
   WHEN "LCU2000" : LighterName := LCU2000;
   WHEN "CWF11"   : LighterName := CWF11;
   WHEN "CWF21"   : LighterName := CWF21;
   WHEN "CWF31"   : LighterName := CWF31;
   WHEN "LSV"     : LighterName := LSV;
```

```
END CASE;

RETURN(LighterName);

END PROCEDURE;


{-------------------------------------------------------------}
PROCEDURE ShipTypeToStr(IN ShipType : ShipTypeType) :
STRING;
{-------------------------------------------------------------}

VAR
Str : STRING;

BEGIN

CASE ShipType
   WHEN SSR : Str := "SSR";
   WHEN NSSR : Str := "NSSR";

   OTHERWISE
      Str := "Other...?";
END CASE;

RETURN(Str);

END PROCEDURE;


{-------------------------------------------------------------}
PROCEDURE StrToShipType(IN Str : STRING) : ShipTypeType;
{-------------------------------------------------------------}

VAR
ShipType : ShipTypeType;

BEGIN
```

```
WriteLine("Converting" + Str);

CASE Str
   WHEN "SSR" : ShipType := SSR;
   WHEN "NSSR" : ShipType := NSSR;


 END CASE;

RETURN(ShipType);

END PROCEDURE;
```

```
{-------------------------------------------------------------}
PROCEDURE BooleanToStr(IN Boolean : BOOLEAN) : STRING;
{-------------------------------------------------------------}

VAR
Str : STRING;

BEGIN

CASE Boolean
   WHEN TRUE : Str := "TRUE";
   WHEN FALSE : Str := "FALSE";

   OTHERWISE
      Str := "Other...?";
END CASE;

RETURN(Str);

END PROCEDURE;
```

```
{-------------------------------------------------------------}
PROCEDURE StrToBoolean(IN Str : STRING) : BOOLEAN;
{-------------------------------------------------------------}

VAR
Boolean : BOOLEAN;

BEGIN

WriteLine("Converting" + Str);

CASE Str
   WHEN "TRUE" : Boolean := TRUE;
   WHEN "FALSE" : Boolean := FALSE;

END CASE;

RETURN(Boolean);

END PROCEDURE;


END MODULE.



DEFINITION MODULE Stats;

{-------------------------------------------------------------
Module Name: Stats                    Last Modified: 21 Jul 93
Author:  J. S. Noel
         Lt.    USN

DESCRIPTION:  Defines the Statistics Object.
-------------------------------------------------------------}
FROM Ship    IMPORT RoRoObj;

TYPE
```

```
SpotIdleTimeRecType = RECORD
   Place : STRING;
   Time : REAL;

END RECORD;

GrandMeanBeachSpotIdleTimeArrayType = ARRAY INTEGER OF
                                      SpotIdleRecType;

StatsObj = OBJECT;

   Reps          : INTEGER;
   NumBeaches    : INTEGER;
   MeanTPut      : REAL;
   MeanLCUinBQ   : REAL;
   MeanLSVinBQ   : REAL;
   MeanCWFinBQ   : REAL;

   GrandMeanLCUinBQ : REAL;
   GrandMeanLSVinBQ : REAL;
   GrandMeanCWFinBQ : REAL;
   GrandMeanLCUinSQ : REAL;
   GrandMeanLSVinSQ : REAL;
   GrandMeanCWFinSQ : REAL;

   MeanLCUinSQ : REAL;
   MeanLSVinSQ : REAL;
   MeanCWFinSQ : REAL;

   MeanShipLCUSpotIdle : REAL;
   MeanShipCWFSpotIdle : REAL;
   MeanShipLoLoSpotIdle : REAL;

   GrandMeanShipLCUSpotIdle : REAL;
   GrandMeanShipCWFSpotIdle : REAL;
   GrandMeanShipLoLoSpotIdle : REAL;

   MeanBeachSpotIdle : REAL;
   GrandMeanBeachSpotIdleTime :
                        GrandMeanBeachSpotIdleTimeArrayType;
```

```
   ASK METHOD ObjInit;
   ASK METHOD DumpStats(IN OffloadTime : REAL);

END OBJECT;

VAR

Stats : StatsObj;

END MODULE.



IMPLEMENTATION MODULE Stats;

{-------------------------------------------------------------
Module Name: Stats                    Last Modified: 21 Jul 93
Author:   J. S. Noel
          Lt.    USN

DESCRIPTION:  Implements the Statistics Object.
-------------------------------------------------------------}
FROM Lighter   IMPORT LighterObj, ALL LighterNameType;
FROM SLCP      IMPORT WaitForShipQueue;
FROM Ship      IMPORT RoRoObj;
FROM Beach     IMPORT BeachObj, ALL BeachType;
FROM Builder   IMPORT BeachBuilder, LighterBuilder,
                      ShipBuilder;
FROM RepMngr   IMPORT RepManager;
FROM Global    IMPORT ALL SpotType;
FROM Convert   IMPORT BeachTypeToStr;
FROM WriteLine IMPORT WriteLine, WriteLineA, WriteLineB,
                      WriteLineC, WriteLineD;
```

```
TYPE

OBJECT StatsObj;

{-------------------------------------------------------------}
ASK METHOD ObjInit;
{-------------------------------------------------------------}

VAR

k : INTEGER;
Beach : BeachObj;
Rec : SpotIdleRecType;

BEGIN

Reps := 0;
MeanTPut := 0.0;

GrandMeanLCUinBQ := 0.0;
GrandMeanLSVinBQ := 0.0;
GrandMeanCWFinBQ := 0.0;
GrandMeanLCUinSQ := 0.0;
GrandMeanLSVinSQ := 0.0;
GrandMeanCWFinSQ := 0.0;

GrandMeanShipLCUSpotIdle := 0.0;
GrandMeanShipCWFSpotIdle := 0.0;
GrandMeanShipLoLoSpotIdle := 0.0;

NumBeaches := ASK BeachBuilder numberIn;
NEW(GrandMeanBeachSpotIdleTime, 1.. NumBeaches);

k :=0;
Beach := ASK BeachBuilder First();
WHILE Beach <> NILOBJ

   k := k + 1;
```

```
    NEW(Rec);
    Rec.Place := ASK Beach BeachID;
    Rec.Time := 0.0;
    GrandMeanBeachSpotIdleTime[k] := Rec;

Beach := ASK BeachBuilder Next(Beach);
END WHILE;

END METHOD;


{-------------------------------------------------------------}
ASK METHOD DumpStats(IN OffloadTime : REAL);
{-------------------------------------------------------------}

VAR

Lighter      : LighterObj;
JunkLighter  : LighterObj;
Ship         : RoRoObj;
Beach        : BeachObj;
i            : INTEGER;
LighterType  : LighterNameType;
NameOfShip   : STRING;
TypeOfBeach  : BeachType;
IDOfBeach    : STRING;

CountLCU     : INTEGER;
CountLSV     : INTEGER;
CountCWF     : INTEGER;
CountLoLo    : INTEGER;

TimeLCUinBQ  : REAL;
TimeLSVinBQ  : REAL;
TimeCWFinBQ  : REAL;

TimeLCUinSQ  : REAL;
TimeLSVinSQ  : REAL;
TimeCWFinSQ  : REAL;
```

```
ShipLCUSpotIdle : REAL;
ShipCWFSpotIdle : REAL;
ShipLoLoSpotIdle : REAL;

BeachSpotIdle : REAL;


BEGIN

WriteLine("Entering Stats mod, DumpStats method.
OffloadTime = " +
         REALTOSTR(OffloadTime));

CountLCU     :=0;
CountLSV     :=0;
CountCWF     :=0;
CountLoLo    :=0;

TimeLCUinBQ := 0.0;
TimeLSVinBQ := 0.0;
TimeCWFinBQ := 0.0;

TimeLCUinSQ := 0.0;
TimeLSVinSQ := 0.0;
TimeCWFinSQ := 0.0;

MeanLCUinBQ := 0.0;
MeanLSVinBQ := 0.0;
MeanCWFinBQ := 0.0;

MeanLCUinSQ := 0.0;
MeanLSVinSQ := 0.0;
MeanCWFinSQ := 0.0;

ShipLCUSpotIdle := 0.0;
ShipCWFSpotIdle := 0.0;
ShipLoLoSpotIdle := 0.0;

MeanShipLCUSpotIdle := 0.0;
MeanShipCWFSpotIdle := 0.0;
```

```
MeanShipLoLoSpotIdle := 0.0;

BeachSpotIdle := 0.0;
MeanBeachSpotIdle := 0.0;


Reps := Reps + 1;
MeanTPut := MeanTPut + OffloadTime;

   Lighter := ASK LighterBuilder First();
   WHILE Lighter <> NILOBJ

        LighterType := ASK Lighter LighterTypeName;

        IF ( LighterType = LCU1466) OR (LighterType =
            LCU1610) OR (LighterType = LCU2000)

           TimeLCUinSQ := TimeLCUinSQ + ASK Lighter
                          TimeInShipQueue;
           TimeLCUinBQ := TimeLCUinBQ + ASK Lighter
                          TimeInBeachQueue;
           CountLCU    := CountLCU + 1;

        ELSIF (ASK Lighter LighterTypeName = LSV )
           TimeLSVinSQ := TimeLSVinSQ + ASK Lighter
                          TimeInShipQueue;
           TimeLSVinBQ := TimeLSVinBQ + ASK Lighter
                          TimeInBeachQueue;
           CountLSV    := CountLSV + 1;

        ELSE
           TimeCWFinSQ := TimeCWFinSQ + ASK Lighter
                          TimeInShipQueue;
           TimeCWFinBQ := TimeCWFinBQ + ASK Lighter
                          TimeInBeachQueue;
           CountCWF    := CountCWF + 1;

        END IF;

   Lighter := ASK LighterBuilder Next(Lighter);
```

```
   END WHILE;

MeanLCUinBQ := TimeLCUinBQ / FLOAT(CountLCU);
MeanLSVinBQ := TimeLSVinBQ / FLOAT(CountLSV);
MeanCWFinBQ := TimeCWFinBQ / FLOAT(CountCWF);

MeanLCUinSQ := TimeLCUinSQ / FLOAT(CountLCU);
MeanLSVinSQ := TimeLSVinSQ / FLOAT(CountLSV);
MeanCWFinSQ := TimeCWFinSQ / FLOAT(CountCWF);

GrandMeanLCUinBQ := GrandMeanLCUinBQ + MeanLCUinBQ;
GrandMeanLSVinBQ := GrandMeanLSVinBQ + MeanLSVinBQ;
GrandMeanCWFinBQ := GrandMeanCWFinBQ + MeanCWFinBQ;
GrandMeanLCUinSQ := GrandMeanLCUinSQ + MeanLCUinSQ;
GrandMeanLSVinSQ := GrandMeanLSVinSQ + MeanLSVinSQ;
GrandMeanCWFinSQ := GrandMeanCWFinSQ + MeanCWFinSQ;

CountLCU    :=0;
CountCWF    :=0;
CountLoLo    :=0;


{ Determine ship spot stats }

IF ( Reps = 1 )

WriteLineC("  ");
WriteLineC("  ");
WriteLineC("---------Idle Ship Spot TimeStats-----------");
WriteLineC("  ");
WriteLineC("  ");
WriteLineC("Rep#   Ship Name     Mean LCU        Mean CWF
          Mean LoLo");

END IF;


Ship := ASK ShipBuilder First();
WHILE Ship <> NILOBJ
```

```
   NameOfShip := ASK Ship ShipName;

   FOR i := 1 TO (ASK Ship NumSpots)
      IF ( ASK Ship ShipSpot[i].SpotClassification = LCU )
         ShipLCUSpotIdle := ASK Ship
                                ShipSpot[i].TotalIdleTime;
         CountLCU    := CountLSV + 1;

      ELSIF( ASK Ship ShipSpot[i].SpotClassification = CWF )
        ShipCWFSpotIdle := ASK Ship
                               ShipSpot[i].TotalIdleTime;
        CountCWF    := CountCWF + 1;

      ELSE
         ShipLoLoSpotIdle := ASK Ship
                                 ShipSpot[i].TotalIdleTime;
         CountLoLo    := CountLoLo + 1;

      END IF;

   END FOR;

MeanShipLCUSpotIdle := ShipLCUSpotIdle /FLOAT(CountLCU);
MeanShipCWFSpotIdle := ShipCWFSpotIdle / FLOAT(CountCWF);
MeanShipLoLoSpotIdle := ShipLoLoSpotIdle / FLOAT(CountLoLo);

GrandMeanShipLCUSpotIdle := GrandMeanShipLCUSpotIdle
                            + MeanShipLCUSpotIdle;
GrandMeanShipCWFSpotIdle := GrandMeanShipCWFSpotIdle
                            + MeanShipCWFSpotIdle;
GrandMeanShipLoLoSpotIdle := GrandMeanShipLoLoSpotIdle
                             + MeanShipLoLoSpotIdle;

WriteLineC(INTTOSTR(Reps) + "       " + NameOfShip +
           "       " + REALTOSTR(MeanShipLCUSpotIdle) +
           "       " + REALTOSTR(MeanShipCWFSpotIdle) +
           "       " + REALTOSTR(MeanShipLoLoSpotIdle));
```

```
Ship := ASK ShipBuilder Next(Ship);

END WHILE;


{ Determine Beach spot stats }

IF ( Reps = 1 )

WriteLineD("  ");
WriteLineD("  ");
WriteLineD("----------Idle Beach Spot TimeStats----------");
WriteLineD("  ");
WriteLineD("  ");
WriteLineD("Rep#   Beach Make          Beach ID         Mean
            Idle Time");

END IF;

k := 0;
Beach := ASK BeachBuilder First();
WHILE Beach <> NILOBJ

   k := k + 1;
   TypeOfBeach := ASK Beach BeachMake;
   IDOfBeach := ASK Beach BeachID;

   FOR i := 1 TO (ASK Beach NumSpots)

      BeachSpotIdle := BeachSpotIdle + ASK Beach
                       BeachSpot[i].TotalIdleTime;

   END FOR;

MeanBeachSpotIdle := BeachSpotIdle / FLOAT(ASK Beach
                                           NumSpots);

WriteLineD(INTTOSTR(Reps) + "       " +
           BeachTypeToStr(TypeOfBeach) + "       " +
```

```
            IDOfBeach + "       " +
            REALTOSTR(MeanBeachSpotIdle));

GrandMeanBeachSpotIdleTime[k].Time :=
   GrandMeanBeachSpotIdleTime[k].Time + MeanBeachSpotIdle;

BeachSpotIdle := 0.0;
Beach := ASK BeachBuilder Next(Beach);

END WHILE;


IF ( Reps = 1 )

WriteLineA("  ");
WriteLineA("  ");
WriteLineA("--------------Throughput Stats---------------");
WriteLineA("  ");
WriteLineA("  ");
WriteLineA("Rep#    Total Time");

END IF;

WriteLineA(INTTOSTR(Reps) + "       " +
            REALTOSTR(OffloadTime));


IF ( Reps = 1 )

WriteLineB("  ");
WriteLineB("  ");
WriteLineB("-------------Time in Queue Stats------------");
WriteLineB("  ");
WriteLineB("  ");
WriteLineB("Rep#     Mean LCU        Mean LSV        Mean
CWF        Mean LCU     Mean LSV     Mean CWF");
WriteLineB("           Ship            Ship            Ship
            Beach        Beach        Beach");

END IF;
```

```
WriteLineB(INTTOSTR(Reps) + "       " +
           REALTOSTR(MeanLCUinSQ) +
           "      " + REALTOSTR(MeanLSVinSQ) + "       " +
           REALTOSTR(MeanCWFinSQ) +
           "       " + REALTOSTR(MeanLCUinBQ) + "     " +
           REALTOSTR(MeanLSVinBQ) +
           "    " + REALTOSTR(MeanCWFinBQ));



IF ( ASK RepManager OutputToScreen )


OUTPUT;
OUTPUT;
OUTPUT("---------------Throughput Stats-----------------");
OUTPUT;
OUTPUT;
END IF;

OUTPUT;
OUTPUT("Rep#    Total Time");
OUTPUT(INTTOSTR(Reps) + "       " + REALTOSTR(OffloadTime));
OUTPUT;
OUTPUT;

{ If more than one ship, must cycle through list to
  determine if  LastLoad = T for each. If true, and RepMngr
  is Done, then simulation is truely finished and Stats can
  be dumped. If any ships still have vehicles, then only the
  current ship can be reset and must ask SLCP to GetLighter.
  GetLighter must be modified to cycle through all ships
  in list when CheckSpots is fired. }

IF (ASK RepManager Done)

WriteLineA("  ");
WriteLineA(" Total number of reps completed = " +
           INTTOSTR(Reps));
```

```
WriteLineB("  ");
WriteLineB(" Total number of reps completed = " +
           INTTOSTR(Reps));
WriteLineC("  ");
WriteLineC(" Total number of reps completed = " +
           INTTOSTR(Reps));
WriteLineD("  ");
WriteLineD(" Total number of reps completed = " +
           INTTOSTR(Reps));
WriteLineD("  ");

MeanTPut := MeanTPut / FLOAT(Reps);

GrandMeanLCUinBQ := GrandMeanLCUinBQ / FLOAT(Reps);
GrandMeanLSVinBQ := GrandMeanLSVinBQ / FLOAT(Reps);
GrandMeanCWFinBQ := GrandMeanCWFinBQ / FLOAT(Reps);
GrandMeanLCUinSQ := GrandMeanLCUinSQ / FLOAT(Reps);
GrandMeanLSVinSQ := GrandMeanLSVinSQ / FLOAT(Reps);
GrandMeanCWFinSQ := GrandMeanCWFinSQ / FLOAT(Reps);

GrandMeanShipLCUSpotIdle := GrandMeanShipLCUSpotIdle
                            / FLOAT(Reps);
GrandMeanShipCWFSpotIdle := GrandMeanShipCWFSpotIdle
                            / FLOAT(Reps);
GrandMeanShipLoLoSpotIdle := GrandMeanShipLoLoSpotIdle
                             / FLOAT(Reps);

FOR i := 1 TO NumBeaches
   GrandMeanBeachSpotIdleTime[i].Time :=
   GrandMeanBeachSpotIdleTime[i].Time / FLOAT(Reps);

   WriteLineD("Grand Mean For Beach Spot Idle Time = "
              + REALTOSTR(GrandMeanBeachSpotIdleTime[i].Time)
              + " For Beach ID = "
              + GrandMeanBeachSpotIdleTime[i].Place);

END FOR;

WriteLineA("Mean Throughput Time = " + REALTOSTR(MeanTPut));
```

```
WriteLineB("Grand Mean For LCU in Beach Queue = "
          + REALTOSTR(GrandMeanLCUinBQ));
WriteLineB("Grand Mean For LSV in Beach Queue = "
          + REALTOSTR(GrandMeanLSVinBQ));
WriteLineB("Grand Mean For CWF in Beach Queue = "
          + REALTOSTR(GrandMeanCWFinBQ));
WriteLineB("Grand Mean For LCU in Ship Queue = "
          + REALTOSTR(GrandMeanLCUinSQ));
WriteLineB("Grand Mean For LSV in Ship Queue = "
          + REALTOSTR(GrandMeanLSVinSQ));
WriteLineB("Grand Mean For CWF in Ship Queue = "
          + REALTOSTR(GrandMeanCWFinSQ));

WriteLineC("Grand Mean For LCU Ship Spot Idle Time = "
          + REALTOSTR(GrandMeanShipLCUSpotIdle));
WriteLineC("Grand Mean For CWF Ship Spot Idle Time = "
          + REALTOSTR(GrandMeanShipCWFSpotIdle));
WriteLineC("Grand Mean For LoLo Ship Spot Idle Time = "
          + REALTOSTR(GrandMeanShipLoLoSpotIdle));

OUTPUT("Mean Throughput Time = " + REALTOSTR(MeanTPut));
OUTPUT;
OUTPUT;


ELSE

   OffloadTime := 0.0;

 {Reset all lighters}

   Lighter := ASK LighterBuilder First();
   WHILE Lighter <> NILOBJ
      ASK Lighter TO ResetLighterStats;

   Lighter := ASK LighterBuilder Next(Lighter);
   END WHILE;

 {Reset the RoRo}
```

```
   Ship := ASK ShipBuilder First();
   WHILE Ship <> NILOBJ

   ASK Ship TO ResetShipStats;

   Ship := ASK ShipBuilder Next(Ship);
   END WHILE;

   ASK SLCP TO ResetSLCP;

 {Reset all beaches}

   Beach := ASK BeachBuilder First();
   WHILE Beach <> NILOBJ
      ASK Beach TO ResetBeachStats;
   Beach := ASK BeachBuilder Next(Beach);
   END WHILE;


END IF;

END METHOD;

END OBJECT;

END MODULE.
DEFINITION MODULE WriteLine;

{-------------------------------------------------------------
Module Name: WriteLine                Last Modified: 20 Jul 93
Author:  M. Bailey                    Modified by: J. S. Noel
         Prof.   NPGS

DESCRIPTION:  Defines the WriteLine procedure for output to
the file "Sim.Out".
-------------------------------------------------------------}

PROCEDURE WriteLine(IN String : STRING);
PROCEDURE WriteLineA(IN String : STRING);
PROCEDURE WriteLineB(IN String : STRING);
```

```
PROCEDURE WriteLineC(IN String : STRING);
PROCEDURE WriteLineD(IN String : STRING);

END MODULE.


IMPLEMENTATION MODULE WriteLine;

{-------------------------------------------------------------
Module Name: WriteLine                  Last Modified: 20 Jul 93
Author:  M. Bailey                      Modified by: J. S. Noel
         Prof.   NPGS

DESCRIPTION:  Implements the WriteLine procedure for output
to the file "Sim.Out".
-------------------------------------------------------------}

FROM IOMod IMPORT FileUseType(Output);
FROM IOMod IMPORT StreamObj;
FROM UtilMod IMPORT DateTime;


VAR
DT : STRING;
TraceStream : StreamObj;
TraceStreamA : StreamObj;
TraceStreamB : StreamObj;
TraceStreamC : StreamObj;
TraceStreamD : StreamObj;

{-------------------------------------------------------------}
PROCEDURE WriteLine(IN String : STRING);
{-------------------------------------------------------------}

BEGIN
IF (TraceStream = NILOBJ)
   NEW(TraceStream);
   ASK TraceStream TO Open("sim.out", Output);
   DateTime(DT);
```

```
   ASK TraceStream TO WriteString(DT);
   ASK TraceStream TO WriteLn;
   ASK TraceStream TO WriteLn;
END IF;

ASK TraceStream TO WriteString(String);
ASK TraceStream TO WriteLn;

END PROCEDURE;


{-------------------------------------------------------------}
PROCEDURE WriteLineA(IN String : STRING);
{-------------------------------------------------------------}

BEGIN
IF (TraceStreamA = NILOBJ)
   NEW(TraceStreamA);
   ASK TraceStreamA TO Open("Total.out", Output);
   DateTime(DT);
   ASK TraceStreamA TO WriteString(DT);
   ASK TraceStreamA TO WriteLn;
   ASK TraceStreamA TO WriteLn;
END IF;

ASK TraceStreamA TO WriteString(String);
ASK TraceStreamA TO WriteLn;

END PROCEDURE;


{-------------------------------------------------------------}
PROCEDURE WriteLineB(IN String : STRING);
{-------------------------------------------------------------}

BEGIN
IF (TraceStreamB = NILOBJ)
   NEW(TraceStreamB);
   ASK TraceStreamB TO Open("Queue.out", Output);
   DateTime(DT);
```

```
   ASK TraceStreamB TO WriteString(DT);
   ASK TraceStreamB TO WriteLn;
   ASK TraceStreamB TO WriteLn;
END IF;

ASK TraceStreamB TO WriteString(String);
ASK TraceStreamB TO WriteLn;

END PROCEDURE;


{-------------------------------------------------------------}
PROCEDURE WriteLineC(IN String : STRING);
{-------------------------------------------------------------}

BEGIN
IF (TraceStreamC = NILOBJ)
   NEW(TraceStreamC);
   ASK TraceStreamC TO Open("SSpot.out", Output);
   DateTime(DT);
   ASK TraceStreamC TO WriteString(DT);
   ASK TraceStreamC TO WriteLn;
   ASK TraceStreamC TO WriteLn;
END IF;

ASK TraceStreamC TO WriteString(String);
ASK TraceStreamC TO WriteLn;

END PROCEDURE;


{-------------------------------------------------------------}
PROCEDURE WriteLineD(IN String : STRING);
{-------------------------------------------------------------}

BEGIN
IF (TraceStreamD = NILOBJ)
   NEW(TraceStreamD);
   ASK TraceStreamD TO Open("BSpot.out", Output);
   DateTime(DT);
```

```
   ASK TraceStreamD TO WriteString(DT);
   ASK TraceStreamD TO WriteLn;
   ASK TraceStreamD TO WriteLn;
END IF;

ASK TraceStreamD TO WriteString(String);
ASK TraceStreamD TO WriteLn;

END PROCEDURE;


END MODULE.
```

APPENDIX D SAMPLE INPUT FILES

----------BchName.dat----------

```
4       # Number of Beaches in this file/simulation scenario


SouthBeach  -> dummy \\ # Name of Beach.  Must be of STRING.
Admin       -> dummy \\ # Name of Beach.  Must be of STRING.
Army        -> dummy \\ # Name of Beach.  Must be of STRING.
Navy        -> dummy \\ # Name of Beach.  Must be of STRING.
```

This file contains the names of the four beaches in the validation scenario. The format is as follows:

NumBeaches ... The number of beach names to be read.

BeachName -> ... Name of beach.

The format is the same for all input files. The first line in the file contains the number of records to be read. The first line of each record contains the record identifier, such as the beach name, followed by the symbol " -> ". The lines that follow contain the rest of the record. The end of a record is indicated by a double slash " \\ ". Comments are preceded by a " # " symbol.

----------BchType.dat----------

```
4          # Number of Beach records in this file.


SouthBeach    ->  # Beach ID.  Must be STRING.
BareBeach         # Type of Beach.  Must be of BeachType.
2                 # Number of Spots.  Must be INTEGER.
```

```
LCU                 # Spot Type.  Must be SpotType.
T                   # Spot Free.  Must be BOOLEAN.

LCU                 # Spot Type.  Must be SpotType.
T                   # Spot Free.  Must be BOOLEAN.

6.0                 # Distance from SLCP.  Must be REAL.
6.0                 # Distance from RefuelArea.  Must be REAL.
\\

Admin          ->   # Beach ID.  Must be STRING.
FloatingCWPier      # Type of Beach.  Must be of BeachType.
1                   # Number of Spots.  Must be INTEGER.

LCU                 # Spot Type.  Must be SpotType.
T                   # Spot Free.  Must be BOOLEAN.

6.0                 # Distance from SLCP.  Must be REAL.
6.0                 # Distance from RefuelArea.  Must be REAL.
\\

Army           ->   # Beach ID.  Must be STRING.
FloatingCWPier      # Type of Beach.  Must be of BeachType.
2                   # Number of Spots.  Must be INTEGER.

LCU                 # Spot Type.  Must be SpotType.
T                   # Spot Free.  Must be BOOLEAN.

LCU                 # Spot Type.  Must be SpotType.
T                   # Spot Free.  Must be BOOLEAN.

6.0                 # Distance from SLCP.  Must be REAL.
6.0                 # Distance from RefuelArea.  Must be REAL.
\\

Navy           ->   # Beach ID.  Must be STRING.
FloatingCWPier      # Type of Beach.  Must be of BeachType.
1                   # Number of Spots.  Must be INTEGER.

LCU                 # Spot Type.  Must be SpotType.
T                   # Spot Free.  Must be BOOLEAN.
```

```
6.0                 # Distance from SLCP.  Must be REAL.
6.0                 # Distance from RefuelArea.  Must be REAL.
\\


----------------------------LtName.dat----------------------


12      # Number of Lighters in this file/simulation scenario


ALFA     -> dummy \\# ID of Lighter. Must be a unique STRING.
BRAVO    -> dummy \\# ID of Lighter. Must be a unique STRING.
CHARLIE  -> dummy \\# ID of Lighter. Must be a unique STRING.
DELTA    -> dummy \\# ID of Lighter. Must be a unique STRING.
ECHO     -> dummy \\# ID of Lighter. Must be a unique STRING.
FOXTROT  -> dummy \\# ID of Lighter. Must be a unique STRING.
GOLF     -> dummy \\# ID of Lighter. Must be a unique STRING.
HOTEL    -> dummy \\# ID of Lighter. Must be a unique STRING.
INDIA    -> dummy \\# ID of Lighter. Must be a unique STRING.
JULIET   -> dummy \\# ID of Lighter. Must be a unique STRING.
KILO     -> dummy \\# ID of Lighter. Must be a unique STRING.
LIMA     -> dummy \\# ID of Lighter. Must be a unique STRING.


----------------------------LtType.dat----------------------


12           # Number of Lighter records in this file.


ALFA      ->  # ID of Lighter.  Must be a unique STRING type.
LCU2000       # Type of Lighter.  Must be of LighterNameType.
LCU           # Spot Type Required.  Must be SpotType.
12.0          # Max Speed.  Must be REAL.
10.0          # Full Load Speed.  Must be REAL.
10            # Max Load in # of vehicles. Must be INTEGER.
92000.0       # Fuel Capacity in gallons.  Must be REAL.
41.6          # Burn Rate in Gallons per hour.  Must be REAL.
0.25          # Min Fuel expressed in percent.  Must be REAL.
```

```
\\

BRAVO     ->  # ID of Lighter.  Must be a unique STRING type.
LCU2000       # Type of Lighter.  Must be of LighterNameType.
LCU           # Spot Type Required.  Must be SpotType.
12.0          # Max Speed.  Must be REAL.
10.0          # Full Load Speed.  Must be REAL.
10            # Max Load in # of vehicles.  Must be INTEGER.
92000.0       # Fuel Capacity in gallons.  Must be REAL.
41.6          # Burn Rate in Gallons per hour.  Must be REAL.
0.25          # Min Fuel expressed in percent.  Must be REAL.
\\

CHARLIE   ->  # ID of Lighter.  Must be a unique STRING type.
LCU2000       # Type of Lighter.  Must be of LighterNameType.
LCU           # Spot Type Required.  Must be SpotType.
12.0          # Max Speed.  Must be REAL.
10.0          # Full Load Speed.  Must be REAL.
10            # Max Load in # of vehicles.  Must be INTEGER.
92000.0       # Fuel Capacity in gallons.  Must be REAL.
41.6          # Burn Rate in Gallons per hour.  Must be REAL.
0.25          # Min Fuel expressed in percent.  Must be REAL.
\\

DELTA     ->  # ID of Lighter.  Must be a unique STRING type.
LCU2000       # Type of Lighter.  Must be of LighterNameType.
LCU           # Spot Type Required.  Must be SpotType.
12.0          # Max Speed.  Must be REAL.
10.0          # Full Load Speed.  Must be REAL.
10            # Max Load in # of vehicles.  Must be INTEGER.
92000.0       # Fuel Capacity in gallons.  Must be REAL.
41.6          # Burn Rate in Gallons per hour.  Must be REAL.
0.25          # Min Fuel expressed in percent.  Must be REAL.
\\

ECHO      ->  # ID of Lighter.  Must be a unique STRING type.
LCU2000       # Type of Lighter.  Must be of LighterNameType.
LCU           # Spot Type Required.  Must be SpotType.
12.0          # Max Speed.  Must be REAL.
10.0          # Full Load Speed.  Must be REAL.
10            # Max Load in # of vehicles.  Must be INTEGER.
92000.0       # Fuel Capacity in gallons.  Must be REAL.
```

```
41.6          # Burn Rate in Gallons per hour.  Must be REAL.
0.25          # Min Fuel expressed in percent.  Must be REAL.
\\


FOXTROT   ->  # ID of Lighter.  Must be a unique STRING type.
LCU1610       # Type of Lighter.  Must be of LighterNameType.
LCU           # Spot Type Required.  Must be SpotType.
12.0          # Max Speed.  Must be REAL.
10.0          # Full Load Speed.  Must be REAL.
4             # Max Load in # of vehicles.  Must be INTEGER.
3290.0        # Fuel Capacity in gallons.  Must be REAL.
36.0          # Burn Rate in Gallons per hour.  Must be REAL.
0.25          # Min Fuel expressed in percent.  Must be REAL.
\\

GOLF      -> # ID of Lighter.  Must be a unique STRING type.
LCU1610       # Type of Lighter.  Must be of LighterNameType.
LCU           # Spot Type Required.  Must be SpotType.
12.0          # Max Speed.  Must be REAL.
10.0          # Full Load Speed.  Must be REAL.
3             # Max Load in # of vehicles.  Must be INTEGER.
3290.0        # Fuel Capacity in gallons.  Must be REAL.
36.0          # Burn Rate in Gallons per hour.  Must be REAL.
0.25          # Min Fuel expressed in percent.  Must be REAL.
\\

HOTEL     ->  # ID of Lighter.  Must be a unique STRING type.
LSV           # Type of Lighter.  Must be of LighterNameType.
LCU           # Spot Type Required.  Must be SpotType.
10.6          # Max Speed.  Must be REAL.


10.0          # Full Load Speed.  Must be REAL.
25            # Max Load in # of vehicles.  Must be INTEGER.
165000.0      # Fuel Capacity in gallons.  Must be REAL.
145.8         # Burn Rate in Gallons per hour.  Must be REAL.
0.25          # Min Fuel expressed in percent.  Must be REAL.
\\

INDIA     ->  # ID of Lighter.  Must be a unique STRING type.
CWF31         # Type of Lighter.  Must be of LighterNameType.
```

```
CWF             # Spot Type Required.  Must be SpotType.
6.0             # Max Speed.  Must be REAL.
3.0             # Full Load Speed.  Must be REAL.
15              # Max Load in # of vehicles.  Must be INTEGER.
1000.0          # Fuel Capacity in gallons.  Must be REAL.
20.8            # Burn Rate in Gallons per hour.  Must be REAL.
0.25            # Min Fuel expressed in percent.  Must be REAL.
\\

JULIET    ->    # ID of Lighter.  Must be a unique STRING type.
CWF31           # Type of Lighter.  Must be of LighterNameType.
CWF             # Spot Type Required.  Must be SpotType.
6.0             # Max Speed.  Must be REAL.
3.0             # Full Load Speed.  Must be REAL.
15              # Max Load in # of vehicles.  Must be INTEGER.
1000.0          # Fuel Capacity in gallons.  Must be REAL.
20.8            # Burn Rate in Gallons per hour.  Must be REAL.
0.25            # Min Fuel expressed in percent.  Must be REAL.
\\

KILO      ->    # ID of Lighter.  Must be a unique STRING type.
CWF31           # Type of Lighter.  Must be of LighterNameType.
CWF             # Spot Type Required.  Must be SpotType.
6.0             # Max Speed.  Must be REAL.
3.0             # Full Load Speed.  Must be REAL.
15              # Max Load in # of vehicles.  Must be INTEGER.
1000.0          # Fuel Capacity in gallons.  Must be REAL.
20.8            # Burn Rate in Gallons per hour.  Must be REAL.
0.25            # Min Fuel expressed in percent.  Must be REAL.
\\

LIMA      ->    # ID of Lighter.  Must be a unique STRING type.
CWF31           # Type of Lighter.  Must be of LighterNameType.
CWF             # Spot Type Required.  Must be SpotType.
6.0             # Max Speed.  Must be REAL.
3.0             # Full Load Speed.  Must be REAL.
15              # Max Load in # of vehicles.  Must be INTEGER.
1000.0          # Fuel Capacity in gallons.  Must be REAL.
20.8            # Burn Rate in Gallons per hour.  Must be REAL.
0.25            # Min Fuel expressed in percent.  Must be REAL.
\\
```

Notes: a. LighterNameType = (LCU1466, LCU1610. LCU2000, CWF11, CWF21, CWF31, LSV).

These are the only options for LighterNameType.

b. SpotType = (LCU, CWF, LoLo). This refers to the type of spot required for mooring by a lighter. An LSV uses an LCU spot for mooring to the RRDF, so this would be indicated in this field.

----------RFAName.dat----------

```
1  # Number of Refuel Areas in this file/simulation scenario

FuelDepot    -> dummy \\ # Name of Refuel Area.  Must be of
                         STRING type.
```

----------RFAType.dat----------

```
1          # Number of Refuel Area records in this file.

FuelDepot      ->       # Name of Refuel Area.  Must be of
STRING type.
2                       # Number of spots.  Must be INTEGER.
6.0                     # Distance from Ship.  Must be REAL.

                        # For each spot there must be a data
                          set.  For 2 spots, we need two
                          BOOLEAN expressions in the following
                          two fields.  If there were three
                          spots, There would be three
                         "SpotFree" fields in this record.

T                       # Spot Free.  Must be BOOLEAN.

T                       # Spot Free.  Must be BOOLEAN.
```

```
3500.0                  # Fuel pump rate in gallons per hour.
                          Must be REAL.
\\


--------------------------ShpName.dat----------------------


1       # Number of Ships in this file/simulation scenario


Belatrix  -> dummy \\ # Name of Ship.  Must be of STRING
                        type.


--------------------------ShpType.dat----------------------


1          # Number of Ship records in this file.


Belatrix      ->        # Name of Ship.  Must be of STRING
type.
SSR         # Ship Type.  Must be ShipTypeType ( SSR, NSSR ).
6.0                     # Distance from BLCP.  Must be REAL.
3                       # Number of spots.  Must be INTEGER.

LCU                     # Spot Type.  Must be SpotType.
T                       # Spot Free.  Must be BOOLEAN.

CWF                     # Spot Type.  Must be SpotType.
T                       # Spot Free.  Must be BOOLEAN.

LoLo                    # Spot Type.  Must be SpotType.
T                       # Spot Free.  Must be BOOLEAN.

60           # Number of LoLo vehicles.  Must be INTEGER.
0            # Number of RRDF Vehicles.  Must be INTEGER.
834          # Number of Any Spot vehicles.  Must be INTEGER.
\\
```

Ship type is either self-sustaining (SSR), or non-self sustaining (NSSR). These are the only two options.

For each spot in the record, two fields must appear:

1. SpotType, must be of type SpotType which is either LCU, CWF, or LoLo. These are the only options for this field.

2. SpotFree, indicates the status of the spot. normally the spot would be empty at the start of a simulation run, thus, indicated by a "T" for TRUE.

The number of vehicles on the ship can be partitioned into three sets. For example, in the set above there are a total of 894 vehicles onboard. Of those, 60 must be offloaded from the LoLo spot, zero are may be removed from the RRDF only, and 834 can be removed from any spot, or the first spot available.

APPENDIX E SAMPLE OUTPUT FILES

This appendix contains a complete set of output files. The sample files below are the results of ten replications of the validation scenario. The files themselves are self explanitory, all times are in minutes.

----------Total.out----------

----------Throughput Stats----------

Rep#	Total Time
1	5795.406696
2	4516.278981
3	5049.667063
4	4212.945852
5	4342.143585
6	4598.425586
7	4306.958894
8	5356.148307
9	4627.004724
10	4096.678710

Total number of reps completed = 10
Mean Throughput Time = 4690.165840

----------BSpot.out----------

----------Idle Beach Spot Time Stats----------

Rep#	Beach Make	Beach ID	Mean Idle Time
1	BareBeach	NorthBeach	1851.511594
1	FloatingCWPier	Admin	2696.831036
1	FloatingCWPier	Army	3044.404815

1	FloatingCWPier	Navy	2706.889934
2	BareBeach	NorthBeach	2126.357278
2	FloatingCWPier	Admin	2633.401745
2	FloatingCWPier	Army	2666.989835
2	FloatingCWPier	Navy	0.000000
3	BareBeach	NorthBeach	2214.400175
3	FloatingCWPier	Admin	2615.639354
3	FloatingCWPier	Army	2678.884777
3	FloatingCWPier	Navy	0.000000
4	BareBeach	NorthBeach	2030.798808
4	FloatingCWPier	Admin	2660.519865
4	FloatingCWPier	Army	2757.632933
4	FloatingCWPier	Navy	0.000000
5	BareBeach	NorthBeach	2056.886539
5	FloatingCWPier	Admin	2778.227232
5	FloatingCWPier	Army	3207.377480
5	FloatingCWPier	Navy	0.000000
6	BareBeach	NorthBeach	2289.250141
6	FloatingCWPier	Admin	2956.250880
6	FloatingCWPier	Army	2864.941143
6	FloatingCWPier	Navy	0.000000
7	BareBeach	NorthBeach	2098.334342
7	FloatingCWPier	Admin	2689.334808
7	FloatingCWPier	Army	3177.835989
7	FloatingCWPier	Navy	0.000000
8	BareBeach	NorthBeach	2368.197230
8	FloatingCWPier	Admin	3099.115844
8	FloatingCWPier	Army	3013.179778
8	FloatingCWPier	Navy	0.000000
9	BareBeach	NorthBeach	2420.594112
9	FloatingCWPier	Admin	2873.591649
9	FloatingCWPier	Army	3006.716735
9	FloatingCWPier	Navy	0.000000
10	BareBeach	NorthBeach	2186.314462
10	FloatingCWPier	Admin	2618.761722
10	FloatingCWPier	Army	3208.601058
10	FloatingCWPier	Navy	0.000000

Total number of reps completed = 10

Grand Mean For Beach Spot Idle Time = 2164.264468 For Beach ID = NorthBeach
Grand Mean For Beach Spot Idle Time = 2762.167413 For Beach ID = Admin
Grand Mean For Beach Spot Idle Time = 2962.656454 For Beach ID = Army
Grand Mean For Beach Spot Idle Time = 270.688993 For Beach ID = Navy

----------SSpot.out----------

----------Idle Ship Spot Time Stats----------

Rep#	Ship Name	Mean LCU	Mean CWF	Mean LoLo
1	Belatrix	1983.547568	2274.516363	1541.758145
2	Belatrix	1512.667104	1215.405897	1984.996518
3	Belatrix	2438.643103	2187.915981	1553.953693
4	Belatrix	2349.907577	1956.123552	1665.716019
5	Belatrix	1469.847130	1351.808132	2062.091047
6	Belatrix	1950.243928	1334.895501	2027.822743
7	Belatrix	820.284977	1445.356970	1481.796293
8	Belatrix	1468.487069	2564.665516	1515.234788
9	Belatrix	2014.772655	1953.579125	1429.237818
10	Belatrix	1474.539864	1584.341778	1938.257845

Total number of reps completed = 10
Grand Mean For LCU Ship Spot Idle Time = 1748.294097
Grand Mean For CWF Ship Spot Idle Time = 1786.860881
Grand Mean For LoLo Ship Spot Idle Time = 1720.086491

----------Queue.out----------

----------Time in Queue Stats----------

Rep#	Mean LCU Ship	Mean LSV Ship	Mean CWF Ship
1	457.868142	630.036495	582.470282
2	716.492200	535.805339	433.802158
3	757.966480	393.900755	466.858381
4	809.126943	285.984057	503.752261
5	902.481843	587.302477	398.722487
6	1099.275997	518.580067	680.860167
7	828.091272	552.074930	456.557370
8	1197.924213	529.680445	465.659741
9	995.634134	716.634971	547.029459
10	983.139182	434.378531	391.346992

Rep#	Mean LCU Beach	Mean LSV Beach	Mean CWF Beach
1	0.000000	0.000000	0.000000
2	0.000000	0.000000	0.000000
3	0.000000	0.000000	0.000000
4	0.000000	0.000000	0.000000
5	0.000000	0.000000	0.000000
6	0.000000	0.000000	0.000000
7	0.000000	0.000000	0.000000
8	0.000000	0.000000	0.000000
9	0.000000	0.000000	0.000000
10	0.000000	0.000000	0.000000

Total number of reps completed = 10
Grand Mean For LCU in Beach Queue = 0.000000
Grand Mean For LSV in Beach Queue = 0.000000
Grand Mean For CWF in Beach Queue = 0.000000
Grand Mean For LCU in Ship Queue = 874.800041
Grand Mean For LSV in Ship Queue = 518.437807
Grand Mean For CWF in Ship Queue = 492.705930

INITIAL DISTRIBUTION LIST

1.	Defense Logistics Studies Information Exchange U.S. Army Logistics Management Center Fort Lee, VA 23801	2
2.	Defense Technical Information Center Cameron Station Alexandria, VA 22304-6145	2
3.	Deputy Chief of Naval Operations (Logistics) N-402D Washington, DC 20350	1
4.	ATTN: Captain Steve Christy DoD Joint Test Directorate JLOTS III Bunker 101 Fort Story, VA 23459	1
5.	Library, Code 0142 Naval Postgraduate School Monterey, CA 93943-5002	2
6.	Professor William G. Kemple, Code OR/KE Naval Postgraduate School Monterey, CA 93943-5002	1
7.	Professor Keebom Kang, Code AS/KK Naval Postgraduate School Monterey, CA 93943-5002	1
8.	Professor David Schrady, Code OR/SO Naval Postgraduate School Monterey, CA 93943-5002	1

9. LPEILD 1
J-4, The Joint Staff
Pentagon
Washington DC, 20318-4000

10. Mobility Division 1
J-4, The Joint Staff
Pentagon
Washington DC, 20318-4000

11. Lieutenant Jack S. Noel 2
Department Head School, Class 131
Newport, RI 02841